RL-TR-94-197
Final Technical Report
September 1995

NEURAL NETWORK APPROACH TO OPTIMAL FILTERING

Maryland Technology Corporation

James T. Lo

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

Rome Laboratory
Air Force Materiel Command
Griffiss Air Force Base, New York

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-94-197 has been reviewed and is approved for publication.

APPROVED: Eric C. Jones

ERIC C. JONES
Project Engineer

FOR THE COMMANDER: Donald W. Hanson

DONALD W. HANSON
Director of Surveillance & Photonics

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify RL (OCTM) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)

2. REPORT DATE
November 1994

3. REPORT TYPE AND DATES COVERED
Final May 91 - May 94

4. TITLE AND SUBTITLE
NEURAL NETWORK APPROACH TO OPTIMAL FILTERING

5. FUNDING NUMBERS
C - F30602-91-C-0033
PE - 62702F
PR - 4506
TA - ZL
WU - 02

6. AUTHOR(S)
James T. Lo

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Maryland Technology Corporation
10210 Sunway Terrace
Ellicott City MD 21042

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Rome Laboratory (OCTM)
26 Electronic Pky
Griffiss AFB NY 13441-4514

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
RL-TR-94-197

11. SUPPLEMENTARY NOTES
Rome Laboratory Project Engineer: Eric C. Jones/OCTM/(315) 330-3573

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This report describes the results of a study investigating a new approach to the problem of optimal filtering. The approach involves synthesizing realizations of a signal process and a measurement process into a filter for estimating the signal process. The filter is made out of an artificial recurrent neural network (RNN). The synthesis is performed through training RNNs. The weights/parameters of an RNN are determined by minimizing a training criterion. After adequate training, the result is a recursive filter optimal for its given RNN architecture where the lagged feedbacks carry the optimal conditional statistics. If an appropriate RNN paradigm and estimation error criterion are selected, the filter of such a paradigm is proven to approximate an optimal filter in performance to any desired degree of accuracy, provided the RNN that constitutes the filter is of a sufficient size. Range extenders and reducers are introduced to minimize the required RNN size, especially when the signal and/or measurement processes keep increasing in magnitude. Some schemes for adapting to unknown filtering environments are presented. These schemes yield maximum likelihood estimates of the unknown parameters and minimum variance estimates of the signals.

14. SUBJECT TERMS
Neural network, Neural filter, Recurrent Network, Kalman filter, Extended Kalman filter, Range reducer, Range extender (see reverse)

15. NUMBER OF PAGES
104

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT
UNCLASSIFIED

18. SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

19. SECURITY CLASSIFICATION OF ABSTRACT
UNCLASSIFIED

20. LIMITATION OF ABSTRACT
UL

14. (Cont'd)

Adaptive filter, Minimum variance estimate

NEURAL NETWORK APPROACH TO OPTIMAL FILTERING

JAMES T. LO
MARYLAND TECHNOLOGY CORPORATION

Contents

List of Figures

1. Introduction

In the standard formulation of the problem in the modern theory of optimal filtering, the signal process and measurement process are described by the mathematical/statistical model:

$$\begin{aligned} x(t+1) &= f(x(t),t) + G(x(t),t)\xi(t), \ x(1) = x_1, & (1.1) \\ y(t) &= h(x(t),t) + \varepsilon(t), & (1.2) \end{aligned}$$

where $x(t)$ is an n-dimensional stochastic process; $y(t)$ is an m-dimensional stochastic process; x_1 is a Gaussian random vector, $\xi(t)$ and $\varepsilon(t)$ are respectively n_1-dimensional and m_1-dimensional Gaussian noise processes with zero means; x_1, $\xi(t)$ and $\varepsilon(t)$ have given joint probability distributions; and $f(x,t)$, $G(x,t)$ and $h(x,t)$ are known functions with such appropriate dimensions and properties that (1.1) and (1.2) describe faithfully the evolutions of the signal and measurement. The problem of discrete-time optimal filtering is to design and make a discrete-time dynamic system that inputs $y(t)$ and outputs an estimate $\hat{x}(t)$ of $x(t)$ at each time $t = 1, 2, \cdots, T$, which estimate minimizes a given estimation error criterion. Here T is a positive integer or infinity. The dynamic system is called an optimal filter with respect to the given estimation error criterion. The dynamic state of the optimal filter at a time t_1 must carry the optimal conditional statistics given all the measurements $y(t)$ that have been received up to and including the time t_1 at the time so that at the next time $t_1 + 1$, the optimal filter will receive and process $y(t_1+1)$ using the optimal conditional statistics from t_1, and then produce the optimal estimate $\hat{x}(t_1 + 1)$. The most widely used estimation error criterion is the mean square error criterion, $E[\| x(t) - \hat{x}(t) \|^2]$, where E and $\| \cdot \|$ denote the expectation and the Euclidean norm respectively. The estimate $\hat{x}(t)$ that minimizes this criterion is called the minimum variance estimate or the least-square estimate.

The most commonly used method of treating such a problem is the use of a Kalman filter (KF) or an extended Kalman filter (EKF). A detailed description of the KF and EKF (and some other approximate nonlinear filters) can be found in e.g., [11], and [1]. The KF and EKF have been applied to a wide range of areas including aircraft/ship inertial and aided-inertial navigation, spacecraft orbit determination, satellite attitude estimation, phase array radar tracking, nuclear power plant failure detection, power station control, oceanographic surveying, biomedical engineering, and process control. Many important papers on the application of the KF and EKF can be found in [24].

In the rare cases where f and h are linear functions of $x(t)$ and G does not depend on $x(t)$, the model, (1.1) and (1.2), is called the linear-Gaussian model. If the KF is used for a linear-Gaussian model, the resulting estimate $\hat{x}(t)$ is the minimum variance (or the least-squares) estimate. In most cases, however, the foregoing linearity conditions on f, h and G are not satisfied and the EKF is used. At each time point, the EKF, which is a suboptimal approximate filter, first linearizes f and G at the estimated value of $x(t)$ and linearizes h at the predicted value of $x(t+1)$. Then the EKF uses the KF equations to update the estimated value of $x(t+1)$ and the predicted value of $x(t+2)$ for the new measurement $y(t+1)$. By iterating the linearization and estimation a certain number of times or until convergence at each time point, we have the so-called iterated EKF (IEKF). Since both the EKF and IEKF involve linearization, they are not optimal filters. In fact, when either the random driving term $G(x(t))\xi(t)$ in (1.1) or the random measurement noise $\varepsilon(t)$ in (1.2) has such large variances and covariances that the aforementioned estimated value and predicted value of the signal are not very close to the true signal, and/or when the functions f, G and h are not very smooth, the linearization may be a poor approximation and the EKF as well as IEKF may yield poor estimates or even fail totally.

This shortcoming of the EKF and IEKF has motivated an enormous amount of work on nonlinear filtering in the past thirty years or so. But the results have been disappointing. With very few, if any, exceptions, the nonlinear filtering results have been confined to research papers and textbooks. The EKF and, to a much less extent, the IEKF remain as the standard filters for estimating stochastic signals.

The 30-year failure is believed to be related to the methodology that has been used since R. E. Kalman derived the KF equations. The methodology is analysis. Starting with a mathematical/statistical model, the methodology searches for a solution consisting of analytic formulas and/or equations that describe the structures and determine the parameters of the filter. In the process of searching, deductive reasoning is used and many assumptions are made to make some special cases analytically tractable. In fact, the KF was derived under the assumptions that f and h are linear in $x(t)$, G does not depend on $x(t)$, and $\xi(t)$ and $\varepsilon(t)$ are Gaussian sequences. The model, (1.1) and (1.2), contains such assumptions as the Markov property, Gaussian distribution, and additive measurement noise. When enough additional assumptions are made to derive explicit filter equations, these assumptions are usually so restrictive and/or unrealistic that they prevent the filter equations from much real-world application.

When not enough additional assumptions are made, the analysis involved is so deep and complicated that it leads mostly to mathematical formulas and equations that are not ready for designing or implementing a real filter. This state of the art is reflected in [14] and [15]. In the few cases where the assumptions are not so bad and the explicit filtering algorithms are available, these filtering algorithms involve such an enormous amount of computation that their real-time implementation is prohibitively expensive if not impossible. Some examples of such cases can be found in [2], [16], and [17].

Because of the inherent inaccuracies and frequent failures of the EKF and IEKF and the restrictive and unrealistic assumptions and prohibitive computational requirements of other existing filters, new filters are needed that consistently yield a high degree of estimation accuracy *vis-à-vis* the information contained in the measurements about the signal and that can be applied in a large variety of real-world situations.

Recent years have seen a rapid growth in the development of artificial neural networks (ANNs), which are also known as connectionist models, parallel distributed processors, neuroprocessors, and neurocomputers. Being crude mathematical models of theorized mind and brain activity, ANNs exploit the massively parallel processing and distributed information representation properties that are believed to exist in a brain. A good introduction to ANNs can be found in [7] and [8].

There is a large number of ANN paradigms such as Hopfield networks, high-order networks, counterpropagation networks, bidirectional associative memories, piecewise linear machines, neocognitrons, self-organizing feature maps, adaptive resonance theory networks, Boltzmann machines, multilayer perceptrons (MLPs), MLPs with various feedback structures, other recurrent neural network paradigms, etc. These and other ANN paradigms have been applied to systems control (e.g., [28]), signal processing (e.g., [13]), Speech Processing (e.g., [18]), and others (e.g., [23]).

There are many research articles concerning applications of ANNs, most of which can be found in the foregoing books, journals (e.g., *IEEE Transactions on Neural Networks*, *Neural Networks*, and *Neural Computation*), and Conference proceedings (e.g., *Proceedings of the International Joint Conference on Neural Networks*). Application of one of the aforementioned neural network paradigms to optimal filtering was reported in [25]. The signal and measurement processes considered therein are described by the linear-Gaussian model and the neural network used is a Hopfield network with the neural activation function slightly modified. The connection weights and neuron biases for the network are determined by using the Kalman filter (KF) equations so that when the Hopfield network stabilizes at each time point, the stable state is the minimum variance estimate. The usefulness of the method is very limited, because it can only be applied to the linear-Gaussian model for which the KF equations are available and the weights and biases of the Hopfield network need to be updated in the operation of the Hopfield network by other means using the Kalman filter equations.

During the reporting period of time, a new neural network approach has been developed. As opposed to the analytic methodology used in the conventional filtering theory, the new method is synthetic in nature. The method synthesizes realizations of the signal process and measurement process into a filter, which is made out of an artificial recurrent neural networks (RNN). We shall call this filter a neural filter.

The synthesis is performed through training RNNs using a set of training data consisting of the realizations of the signal and measurement processes. More specifically, the weights and/or parameters of an RNN are determined by minimizing a training criterion by the variation of the weights and/or parameters. The training criterion incorporates all the training data and is a function of the weights and/or parameters of the RNN under training.

After adequate training, the neural filter is a recursive filter optimal for the given RNN architecture with the lagged feedbacks carrying the optimal conditional statistics. If appropriate RNN paradigms and estimation error criteria are selected, the neural filter of such a paradigm is proven to approximate an optimal filter in performance (with respect to the selected estimation error criteria) to any desired degree of accuracy, provided that the RNN that constitutes the neural filter is of sufficiently large size.

If a mathematical model of the signal and measurement processes such as (1.1) and (1.2) is available, the realizations of the signal and measurement processes are generated by computer simulation. Otherwise, these training data can be collected by conducting actual experiments with the signal and measurement processes. Since we do not use a mathematical model to derive formulas and equations, such properties as the Markov property, Gaussian distribution and additive noise are not required of the signal and measurement processes for the present invention to apply. In fact, the new approach applies to virtually any signal process (to be defined in the sequel) and measurement process.

Extensive numerical work was carried out to compare the neural filters and the KF, EKF and IEKF for signal and measurement processes that can be described by the mathematical model (1.1) and (1.2). The numerical results show that the neural filters almost equal the KF in performance for a linear model in both transient and steady states of filtering and always outperform the EKF and IEKF for a nonlinear model, when there is sufficient number of neurons (or processing elements) in the neural filter.

This report is organized as follows. Chapter 2 is concerned with optimal filtering by multilayer perceptrons with interconnected hidden neurons. It is proven that after adequate training, such an RNN can approximate an optimal filter in performance with respect to the mean square error criterion to any desired degree of accuracy, provided that the RNN is of sufficient size. Numerical results are given to illustrate the theory. In Chapter 3, analogous theoretical and numerical results are presented for optimal filtering by multilayer perceptrons with output feedbacks. If the signal and/or measurement processes keep increasing in magnitude, some methods for keeping the sizes of the neural filter and the training data set manageable are proposed in Chapter 4. These methods of using range extenders and reducers are discussed and their numerical results provided therein to illustrate their usefulness. In Chapter 5, adaptive filtering is treated and a surprisingly simple and powerful method is presented. The initial numerical results that have been obtained are extremely encouraging, calling for more research on this topic. A byproduct of the reported work on neural filtering, which is somewhat incongruent with the preceding chapters, is the higher-order differentiation formulas for a multilayer perceptron given in Chapter 6. The higher-order derivatives, which these differentiation formulas calculate recursively, provide interpretations of the training data and suggest possible pruning of the neural network connections and neurons. Chapter 7 is a brief conclusion.

2. Optimal Filtering by Multilayer Perceptron with Interconnected Neurons

This chapter was inspired by the recent resurgence of the study on artificial neural networks, especially the multilayer perceptrons (MLPs) and MLP-based recurrent networks. The synaptic weights of such a network are not evaluated by formulas or equations, but are determined by training the network using the desired input/output pairs[5, 12, 19, 22, 26]. This training is synthesizing. Of course, we do not have to use these neural networks for synthesizing filters. Nevertheless, there are three reasons for using them: First, any measurable function with a compact support can be approximated to any degree of accuracy by an MLP with even a single layer of hidden neurons[3, 6, 9]. Second, the feedbacks in a recurrent network provide the "state" in building a dynamical system. Third, the massively parallel nature of MLPs and MLP-based recurrent networks make them most suitable for real-time filtering.

To simplify our discussion, we will, in this chapter, mainly consider an MLP with a single hidden layer of neurons, whose states are fully[5, 21] or partially fed back to one another. Our results can be easily generalized or extended to many other architectures.

In training a recurrent MLP into a filter, the measurement sequence $y(t)$ are used as the inputs consecutively in time and the signal sequence $\psi(x(t))$ are the corresponding desired outputs. A requirement here that is not needed in the modern theory of optimal filtering is that $x(t)$ and $y(t)$ stay in a compact set. The fullfilment of it is easy to justify in the real world. The model, (1.1) and (1.2), violates this requirement. However, it is only an idealization, whose $x(t)$ and $y(t)$ may stray out of a compact region with arbitrarily small probability provided that the region is sufficiently large.

The training process minimizes the mean square error between the network outputs $\hat{\psi}(x(t))$ and the desired outputs $\psi(x(t))$. Consequently, after adequate training, the estimates $\hat{\psi}(x(t))$ produced by the network are minimum-variance for the given network architecture. It is easy to see that the mean square error between $\hat{\psi}(x(t))$ and $\psi(x(t))$ after adequate training decreases, as the number of hidden neurons in the recurrent MLP increases. We call such a recurrent MLP a neural filter. The main theorem in this paper is that the output $\hat{\psi}(x(t))$ of a neural filter converges to the minimum variance estimate, which is also the conditional expectation, $E[\psi(x(t))|y^t]$, as the number of fully interconnected hidden neurons approaches infinity.

Four numerical examples are thoroughly worked out and reported. Two of them show that even the neural filters with a few hidden neurons consistently outperform the extended Kalman filter and the iterated extended Kalman filter for nonlinear systems of (1.1) and (1.2). The third shows that a neural filter with a sufficient number of neurons is indistinguishable from the Kalman filter for a linear system of (1.1) and (1.2). In the last example, neither the signal nor the measurement can be described by (1.1) or (1.2). Nevertheless, the neural filter works satisfactorily.

A close look at the proof of the main theorem indicates that fully interconnectedness of the hidden layer is not necessary for the convergence to the minimum variance filter. Another architecture is also tested in the numerical examples. The hidden neurons in this architecture are arranged in a ring and each is fed back to its two neighbors after a unit time delay. As it stands, the main theorem does not apply to this architecture. However, in all the four examples, the filtering performance of a recurrent MLP with this architecture is very close to that of a recurrent MLP with fully interconnected hidden neurons. Notice that the total number of connections in the architecture with fully interconnected hidden neurons is much greater than that in the architecture with the ring topology, especially when the number of neurons is large. This raises the question whether a neural filter with ring-connected neurons can approximate the minimum variance filter to any degree of accuracy. The question is under investigation and an answer will soon be reported.

2.1. MLPs with Interconnected Hidden Neurons

A typical recurrent MLP (RMLP) with a single hidden layer of fully interconnected neurons is illustrated in Fig. 2.1. It has 2 input variables $y_i(t)$, 4 hidden neurons, and 2 output variables $\alpha_i(t)$, where t indicates the dependence on time. Denote the weight from the ith input to neuron j by w^1_{ji}, the weight from neuron i to the jth output by w^2_{ji}, and the weight for the lagged feedback from neuron i to neuron j by w^r_{ji}. The activation level $\beta_j(t)$ of and the weighted sum $\eta_j(t)$ in neuron j satisfy

$$\beta_j(t) = a(\eta_j(t)), \tag{2.1}$$

$$\eta_j(t) = w_{j0} + \sum_{i=1}^{m} w^1_{ji} y_i(t) + \sum_{i=1}^{q} w^r_{ji}\beta_i(t-1), \tag{2.2}$$

where a is a monotone increasing neuron activation function, say tanh. The output $\alpha_i(t)$ is then determined by

$$\alpha_i(t) = \sum_{j=1}^{q} w^2_{ij}\beta_j(t), \text{ for } i = 1, \cdots, k, \tag{2.3}$$

where m, q, and k are the number of inputs, neurons, and outputs, respectively. It is assumed in this paper that the activation levels are initialized at $\beta_j(0) = 0$, $j = 1, \cdots, q$.

A typical RMLP with a single hidden layer of ring-connected neurons is illustrated in Fig. 2.2. It has 2 input variables $y_i(t)$, 5 hidden neurons, and 2 output

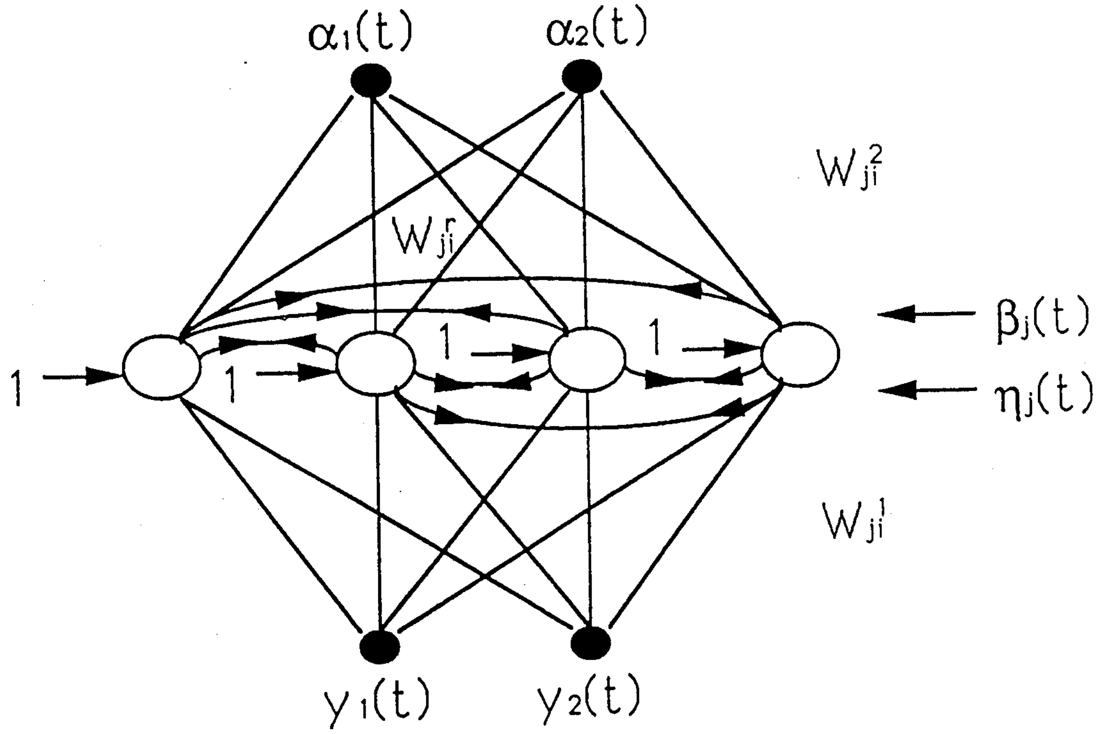


Figure 2.1: A recurrent MLP with fully interconnected hidden neurons

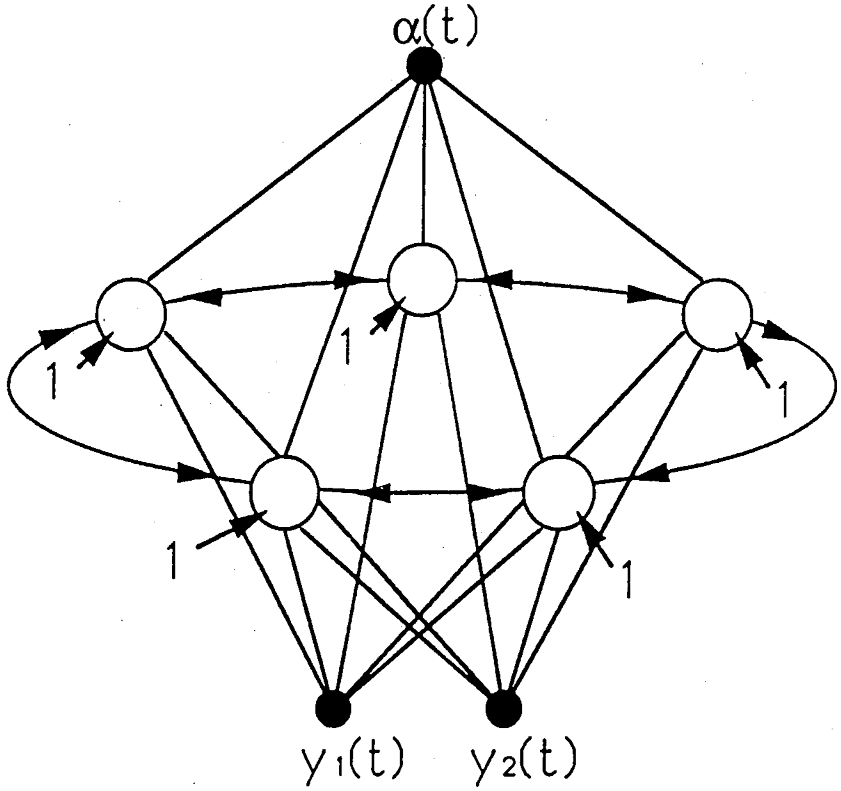


Figure 2.2: A recurrent MLP with ring-connected hidden neurons

variables $\alpha_i(t)$. Using the same symbols as defined above, the RMLP is specified by the equations, (2.1), (2.3), and

$$\begin{aligned}\eta_j(t) &= w_{j0} + \sum_{i=1}^{m} w_{ji}^1 y_i(t) + w_{j\rho(j-1)}^r \beta_{\rho(j-1)}(t-1) \\ &\quad + w_{j\rho(j+1)}^r \beta_{\rho(j+1)}(t-1), \qquad (2.4)\end{aligned}$$

where the function ρ is defined by $\rho(j) = j$, if $1 \leq j \leq q$; $= 1$, if $j = q+1$; $= q$, if $j = 0$. It is also assumed here that $\beta_j(0) = 0$, $j = 1, \cdots, q$.

As discussed in the preceding section, by either computer simulation or experiments, we may have available a finite set of signal/measurement sequences, $(x(t,\omega), y(t,\omega))$, $t = 1, \cdots, T$, $\omega \in S$. It is assumed that the finite set S is a random sample from the sample space Ω and adequately reflects the joint probability distributions of the signal and measurement prosesses $x(t)$ and $y(t)$.

Suppose that $\psi(x(t)) = [\psi_1(x(t)), \cdots, \psi_k(x(t))]^T$ are what we want to estimate and are thus the desired outputs for the actual outputs $\alpha_1(t), \cdots, \alpha_k(t)$. Using the m components of the measurement vector $y(t) = [y_1(t), \cdots, y_m(t)]^T$ as the inputs, the training data consists of the I/O pairs $(y(t,\omega), \psi(x(t,\omega)))$, $t = 1, \cdots, T, \omega \in S$.

The training is to minimize, by the variation of the RMLP weights w, the mean square error

$$C(w) = \frac{1}{T(\#S)} \sum_{\omega \in S} \sum_{t=1}^{T} \sum_{l=1}^{k} (\alpha_l(t,\omega) - \psi_l(x(t,\omega)))^2 \qquad (2.5)$$

where $\#S$ is the number of elements in S and $\alpha_l(t,\omega)$ is the lth actual RMLP output at time t corresponding to the input sequence $y(\tau,\omega)$, $\tau = 1, \cdots, t$. We note here that $\alpha_l(t,\omega)$ is a function of $y(\tau,\omega)$, $\tau = 1, \cdots, t$.

A good training method, which is used in our simulation, is first to evaluate the gradient of the above error function with respect to w by the time-dependent recurrent backpropagation by Werbos[27] and Pearlmutter[20] and then to use it in a conjugate gradient algorithm for optimization.

An adequately trained network through minimizing $C(w)$ will be called a neural filter.

2.2. Convergence to the Minimum Variance Filter

The collection of all k-dimensional random vectors $x = [x_1, \cdots, x_k]^T$ is a Hilbert space L_2 with the norm $E[\|x\|^2] = E[\sum_{l=1}^{k} x_l^2]$, where E denotes the expectation or the average over the sample space Ω with the σ-field $\mathcal{A}$ of events and the probability measure P.

By simple calculation, we have for any Borel-measurable function f of $y^t := \{y(1), \cdots, y(t)\}$,

$$\begin{aligned} C' &:= \frac{1}{T}\sum_{t=1}^{T} E[\|\psi(x(t)) - f(y^t)\|^2] \\ &= \frac{1}{T}\sum_{t=1}^{T}\{E[\|\psi(x(t)) - E[\psi(x(t))|y^t]\|^2] \\ &\quad + E[\|E[\psi(x(t))|y^t] - f(y^t)\|^2]\}, \end{aligned} \tag{2.6}$$

which shows that the conditional expectation $E[\psi(x(t))\ |y^t]$ is the minimum variance estimate of $\psi(x(t))$ given y^t.

Under the assumption that averaging over S is indistinguishable from taking expectation over Ω, we see, by comparing $C(\omega)$ in (2.5) and C' in (2.6), that

$$\begin{aligned} C(w) = {} & \frac{1}{T(\#S)}\sum_{\omega\in S}\sum_{t=1}^{T} \|\psi(x(t,\omega)) - E[\psi(x(t))|y^t(\omega)]\|^2 \\ & + \frac{1}{T(\#S)}\sum_{\omega\in S}\sum_{t=1}^{T} \|\alpha(t,\omega) - E[\psi(x(t))|y^t(\omega)]\|^2 \end{aligned}$$

Training amounts to minimizing the second term above. The objective is, of course, to minimize $(1/T)\sum_{t=1}^{T} E[\|\alpha(t) - \psi(x(t))\|^2]$.

Recall the neural filter architecture described in Sec. 2. We observe that adding a neuron to a hidden layer decreases $\min_w C(w)$. We are now ready to state and prove the main theorem of this paper.

Theorem. Consider the n-dimensional random process $x(t)$ and the m-dimensional random process $y(t)$, $t = 1, \cdots, T$ defined on a probability space $(\Omega, \mathcal{A}, P)$. Assume that the range $\{y(t,\omega)|t = 1, \cdots, T,\ \omega \in \Omega\} \subset R^m$ is compact and ψ is an arbitrary k-dimensional Borel function of $x(t)$ with finite second moments $E[\|\psi(x(t))\|^2]$, $t = 1, \cdots, T$. Let $\alpha(t)$ denote the k-dimensional output at time t of an RMLP which has taken the inputs, $y(1), \cdots, y(t)$, in the given order.

(a). Given $\epsilon > 0$, there exist an RMLP with one hidden layer of fully interconnected neurons such that

$$\frac{1}{T}\sum_{t=1}^{T} E[\|\alpha(t) - E[\psi(x(t))|y^t]\|^2] < \epsilon.$$

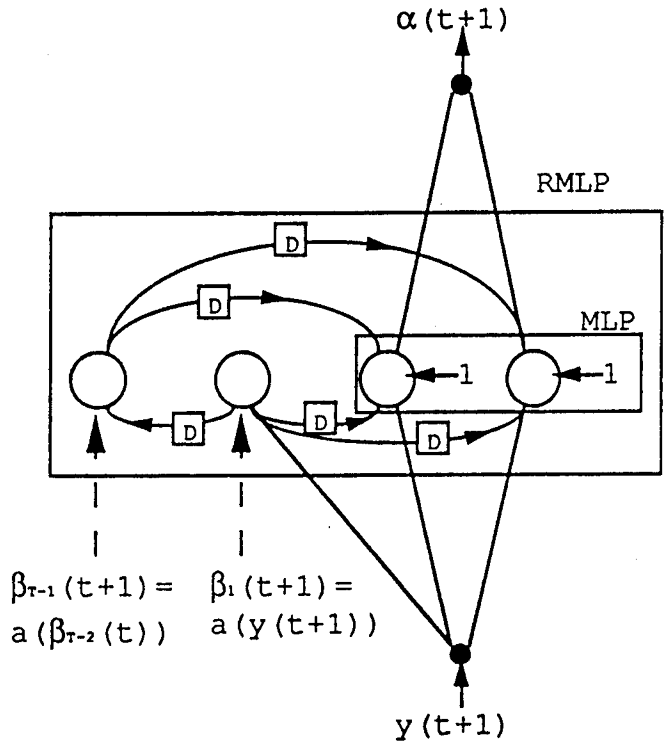


Figure 2.3: A special recurrent MLP used in the proof of the main theorem

(b). If the RMLP has one hidden layer of N neurons fully interconnected and the output $\alpha(t)$ is written as $\alpha(t;N)$ here to indicate its dependency on N, then

$$r(N) := \min_{w} \frac{1}{T} \sum_{t=1}^{T} E[\|\alpha(t,N) - E[\psi(x(t))|y^t]\|^2] \tag{2.7}$$

is monotone decreasing and converges to 0 as N approaches infinity.

Proof. Since (a) is an immediate consequence of (b), we shall only prove (b).

It is obvious that $r(N)$, $N = 1, 2, \cdots$, are all bounded from below by 0. Hence the monotone decreasing sequence $r(N)$ converges and we denote the limit by $r(\infty)$. To prove that $r(\infty) = 0$, it suffices to show that for any $\epsilon > 0$, there is an integer M such that $|r(M)| < \epsilon$. This will be shown in the following for the case in which the measurement process $y(t)$ is scalar-valued, i.e. $m = 1$. The proof for $m > 1$ is similar and omitted.

As illustrated in Fig. 2.3, the first $T - 1$ neurons can be so interconnected that for $t = 0, \cdots, T-1$, $\beta_1(t+1) = a(y(t+1))$ and for $\tau = 2, \cdots, T-1$,

$$\beta_\tau(t+1) = a(\beta_{\tau-1}(t)). \tag{2.8}$$

These neurons are initialized at $\beta_\tau(0) = B$, $\tau = 1, \cdots, T-1$, where B is a point in R^1 outside the compact set $\{y(t,\omega) | t = 1, \cdots, T, \omega \in \Omega\}$. At time $t+1$, the lagged feedbacks from them are

$$\begin{aligned} &(\beta_{T-1}(t), \cdots, \beta_{t+1}(t), \beta_t(t), \beta_{t-1}(t), \cdots, \beta_1(t)) \\ = \;& (B, \cdots, B, a^{\circ t}(y(1)), a^{\circ(t-1)}(y(2)), \cdots, a(y(t))), \end{aligned}$$

where $a^{\circ t}$ denote the t-fold composition $a \circ a \circ \cdots \circ a$ of a. Including both these feedbacks and the input node, we have the T-dimensional vector at time $t+1$,

$$z(t+1) = [\beta_{T-1}(t), \cdots, \beta_1(t), y(t+1)]^T,$$

to use as the inputs to an MLP with one hidden layer.

Recall that $E[\psi(x(t+1)) | a^{\circ t}(y(1)), a^{\circ(t-1)}(y(2)), \cdots, a(y(t)), y(t+1)]$ is a Borel measurable function of $a^{\circ t}(y(1))$, $\cdots$, $a(y(t))$ and $y(t+1)$. Let us denote it by $\lambda_{t+1}(a^{\circ t}(y(1)), \cdots, a(y(t)), y(t+1))$. Now let us consider the mapping $f : R^T \rightarrow R^k$ defined by

$$\begin{aligned} & f(\xi_1, \cdots, \xi_T) \\ = & \left\{ \begin{array}{ll} \lambda_1(\xi_T), & \text{if } \xi_1 = \cdots = \xi_{T-1} = B, \xi_T \neq B \\ & \text{and } \lambda_1(\xi_T) \text{ defined;} \\ \lambda_2(\xi_{T-1}, \xi_T), & \text{if } \xi_1 = \cdots = \xi_{T-2} = B, \xi_{T-1} \neq B, \\ & \xi_T \neq B \text{ and } \lambda_2(\xi_{T-1}, \xi_T) \text{ defined;} \\ \vdots & \\ \lambda_T(\xi_1, \cdots, \xi_T), & \text{if } \xi_1 \neq B, \cdots, \xi_T \neq B \text{ and} \\ & \lambda_T(\xi_1, \cdots, \xi_T) \text{ defined;} \\ 0, & \text{otherwise.} \end{array} \right. \end{aligned}$$

Let us consider also the measures μ_t defined on the Borel sets in R^T by, denoting $d\xi_1 \times \cdots \times d\xi_T$ by $d\xi$,

$$\mu_1(d\xi) = \left\{ \begin{array}{ll} \frac{1}{T} P(y(1) \in d\xi_T), & \text{if } B \in d\xi_t, \text{ for } t < T, and \\ & B \notin d\xi_t, \text{ for } t \not< T, \\ 0, & \text{otherwise;} \end{array} \right.$$

$$\mu_2(d\xi) = \left\{ \begin{array}{l} \frac{1}{T} P(a(y(1)) \in d\xi_{T-1}, y(2) \in d\xi_T), \\ \quad \text{if } B \in d\xi_t, \text{ for } t < T-1, and \\ \quad \text{if } B \notin d\xi_t, \text{ for } t \not< T-1, and \\ 0, \;\text{otherwise;} \end{array} \right.$$

$$\vdots$$

$$\begin{aligned}\mu_T(d\xi) &= \frac{1}{T}P(a^{\circ(T-1)}(y(1)) \in d\xi_1, \cdots, \\ &\quad a(y(T-1)) \in d\xi_{T-1}, y(T) \in d\xi_T).\end{aligned}$$

Notice that $\mu = \mu_1 + \cdots + \mu_T$ is also a mesure on the Borel sets in R^T and that

$$\begin{aligned}&\int \|f(\xi_1, \cdots, \xi_T)\|^2 \mu(d\xi) \\ &= \frac{1}{T}\{\int \|\lambda_1(\xi_T)\|^2 \mu_1(d\xi) \\ &\quad + \int \|\lambda_2(\xi_{T-1}, \xi_T)\|^2 \mu_2(d\xi) \\ &\quad + \cdots \\ &\quad + \int \|\lambda_T(\xi_1, \cdots, \xi_T)\|^2 \mu_T(d\xi)\} \\ &= \frac{1}{T}\{E[\|E[\psi(x(1))|y(1)]\|^2] + \cdots \\ &\quad + E[\|E[\psi(x(T))|a^{\circ(T-1)}(y(1)), \cdots, y(T)]\|^2]\} \\ &\le \frac{1}{T}\{E[\|\psi(x(1))\|^2] + \cdots + E[\|\psi(x(T))\|^2]\} \\ &< \infty.\end{aligned}$$

Since $y(t)$, $t = 1, \cdots, T$, are assumed to have a compact range, it is easy to see that $a^{\circ t}(y(\tau))$, for $t = 0, \cdots, T-1$ and $\tau = 1, \cdots, T$, all have compact ranges. Hence there is a compact set W in R^T such that $\mu(W) = 1$.

Consider now the normed space $L_2(R^T, \mu)$ with the norm defined by $[\int \|g(\xi_1, \cdots, \xi_T)\|^2\ \mu(d\xi)]^{1/2}$ for each k-dimensional function $g \in L_2(R^T, \mu)$. By Corollary 2.2 of the paper [9] by Hornik, Stinchcombe and White, the k-dimensional functions represented by all the MLPs with k output nodes and a single layer of hidden neurons are dense in $L_2(R^T, \mu)$. It then follows that for any $\epsilon > 0$, there is such an MLP that the function $g(\xi_1, \cdots, \xi_T)$ that the MLP represents satisfies

$$\int \|f(\xi_1, \cdots, \xi_T) - g(\xi_1, \cdots, \xi_T)\|^2 \mu(d\xi) < \epsilon.$$

Let us translate the left hand side of this inequality from $L_2(R^T, \mu)$ back to the probability space $(\Omega, \mathcal{A}, P)$:

$$\int \|f(\xi_1, \cdots, \xi_T) - g(\xi_1, \cdots, \xi_T)\|^2 \mu(d\xi)$$

$$
\begin{aligned}
&= \frac{1}{T}\{\int \|\lambda_1(\xi_T) - g(B,\cdots,B,\xi_T)\|^2 \mu_1(d\xi) \\
&+ \int \|\lambda_2(\xi_{T-1},\xi_T) - g(B,\cdots,B,\xi_{T-1},\xi_T)\|^2 \mu_2(d\xi) \\
&+ \cdots \\
&+ \int \|\lambda_T(\xi_1,\cdots,\xi_T) - g(\xi_1,\cdots,\xi_T)\|^2 \mu_T(d\xi)\} \\
&= \frac{1}{T}\sum_{t=1}^{T} E[\|E[\psi(x(t))|y^t] \\
&-g(B,\cdots,B,a^{\circ(t-1)}(y(1)),a^{\circ(t-2)}(y(2)),\cdots,y(t))\|^2]
\end{aligned}
$$

where the last equality holds, because the σ-fields generated by y^t and $\{a^{\circ(t-1)}(y(1)), a^{\circ(t-2)}(y(2)), \cdots, y(t)\}$ are identical due to the monotonicity of a.

We notice that the foregoing approximation MLP, $g(\xi_1,\cdots,\xi_T)$, and the $T-1$ neurons (2.8) discussed earlier form an RMLP with a single hidden layer. The number N of hidden neurons of this RMLP is the sum of that of the MLP and $T-1$. Recalling (2.7), we have

$$
\begin{aligned}
r(N) \le \frac{1}{T}\sum_{t=1}^{T} E[\|E[\psi(x(t))|y^t] - g(B,\cdots,B,& \\
a^{\circ(t-1)}(y(1)),a^{\circ(t-2)}(y(2)),\cdots,y(t))\|^2] &< \epsilon,
\end{aligned}
$$

since $r(N)$ minimizes the error function in (2.7) by the variation of the weights of an RMLP with exactly the same architecture as for the RMLP constructed earlier. This completes the proof. □

In real-world application of a filter, the time interval in which the filter is applied is usually very long, i.e. $T \gg 1$. Having more than $T-1$ neurons is impractical. Nevertheless, the above theorem guarantees you may get arbitrarily close to the minimum variance filter, provided that a sufficient number of neurons are used.

2.3. Differentiating an RMLP with Interconnected Hidden Neurons

In this section, we shall briefly review an efficient method of evaluating the gradient dC/dw, which was derived and called the time-dependent recurrent backpropagation (TDRBP) by Werbos[27] and Pearlmutter[20]. To simplify our discussion, let us consider an RMLP, for $T = 3$, with a single input $y(t)$, a single input $x(t)$, and a hidden layer of q fully interconnected neurons, which are governed by (2.1), (2.2), and (2.3). The training process minimizes, by the variation of w, $C(w)$ in (2.5), which we write as

$$C(w) = \frac{1}{2}\sum_{t=1}^{T}\sum_{\omega\in S}(\alpha(t,\omega) - \xi(t,\omega))^2,$$

where $T(\#S)/2$ is suppressed for simplification.

Before deriving the formulas for $dC(w)/dw$, we observe that $\alpha(t,\omega)$ and $\beta_i(t,\omega)$, $i = 1,\cdots,q$ are functions of w, $y(t,\omega)$, and $\beta_i(t-1,\omega)$, $i = 1,\cdots,q$. By the chain rule of differentiation, we have

$$\begin{aligned}
\frac{dC}{dw} &= \sum_t\sum_\omega \frac{\partial C}{\partial\alpha(t,\omega)}\cdot\frac{d\alpha(t,\omega)}{dw} \\
&= \sum_\omega\{\frac{\partial C}{\partial\alpha(1,\omega)}\cdot\frac{\partial\alpha(1,\omega)}{\partial w} + \sum_{t=2}^{3}\frac{\partial C}{\partial\alpha(t,\omega)}[\sum_i \frac{\partial\alpha(t,\omega)}{\partial\beta_i(t-1,\omega)}\cdot\frac{d\beta_i(t-1,\omega)}{dw} \\
&\quad +\frac{\partial\alpha(t,\omega)}{\partial w}]\} \\
&= \sum_\omega\{\frac{\partial C}{\partial\alpha(1,\omega)}\cdot\frac{\partial\alpha(1,\omega)}{\partial w} + \frac{\partial C}{\partial\alpha(2,\omega)}[\sum_i \frac{\partial\alpha(2,\omega)}{\partial\beta_i(1,\omega)}\cdot\frac{\partial\beta_i(1,\omega)}{\partial w} + \frac{\partial\alpha(2,\omega)}{\partial w}] \\
&\quad +\frac{\partial C}{\partial\alpha(3,\omega)}(\sum_j \frac{\partial\alpha(3,\omega)}{\partial\beta_j(2,\omega)}[\sum_i \frac{\partial\beta_j(2,\omega)}{\partial\beta_i(1,\omega)}\cdot\frac{\partial\beta_i(1,\omega)}{\partial w} + \frac{\partial\beta_j(2,\omega)}{\partial w}] \\
&\quad +\frac{\partial\alpha(3,\omega)}{\partial w})\} \\
&= \sum_\omega\{\sum_t \frac{\partial C}{\partial\alpha(t,\omega)}\cdot\frac{\partial\alpha(t,\omega)}{\partial w} + \sum_i[\frac{\partial C}{\partial\alpha(2,\omega)}\cdot\frac{\partial\alpha(2,\omega)}{\partial\beta_i(1,\omega)} \\
&\quad +\frac{\partial C}{\partial\alpha(3,\omega)}\sum_j \frac{\partial\alpha(3,\omega)}{\partial\beta_j(2,w)}\cdot\frac{\partial\beta_j(2,\omega)}{\partial\beta_i(1,\omega)}]\cdot\frac{\beta_i(1,\omega)}{\partial w} \\
&\quad +\frac{\partial C}{\partial\alpha(3,\omega)}\sum_i \frac{\partial\alpha(3,\omega)}{\partial\beta_i(2,\omega)}\cdot\frac{\partial\beta_i(2,\omega)}{\partial w}\} \\
&= \sum_\omega\{\sum_t \frac{\partial C}{\partial\alpha(t,\omega)}\cdot\frac{\partial\alpha(t,\omega)}{\partial w} + \sum_{t=1}^{T}\sum_i \frac{\partial^+ C}{\partial\beta_i(t,\omega)}\cdot\frac{\partial\beta_i(t,\omega)}{\partial w}\},
\end{aligned} \tag{2.9}$$

where $\partial^+ C/\partial\beta_i(t,\omega)$ is called an ordered partial derivative by Werbos[27] and denotes the partial derivative of C w.r.t. $\beta_i(t,\omega)$ taking into consideration the dependency of $\alpha(t+1,\omega)$ on $\beta_i(t,\omega)$. We notice that the ordered partial derivatives satisfy the recurrent relationship

$$\frac{\partial^+ C}{\partial\beta_i(t,\omega)} = \frac{\partial C}{\partial\alpha(t+1,\omega)} \cdot \frac{\partial\alpha(t+1,\omega)}{\partial\beta_i(t,\omega)} + \sum_j \frac{\partial^+ C}{\partial\beta_j(t+1,\omega)} \cdot \frac{\partial\beta_j(t+1,\omega)}{\partial\beta_i(t,\omega)}, \quad (2.10)$$

with the final condition, $\partial^+ C/\partial\beta_i(T,\omega) = \partial C/\partial\beta_i(T,\omega)$. This relationship is the TDRBP.

The formulas (2.9) and (2.10) constitute a very efficient technique for evaluating the gradient dC/dw. For the numerical examples in the next section, the RMLPs are trained by first applying TDRBP and then optimizing by the conjugate gradient method.

2.4. Numerical Examples

Four examples are worked out. The first two show that both a neural filter with fully-interconnected neurons (NFFN) and that with ring-connected neurons (NFRN) outperform the extended Kalman filter (EKF) and the iterated extended Kalman filter (IEKF) for nonlinear systems described by (1.1) and (1.2). It is also shown that the NFFNs converge rather fast for the example systems. By the main theorem, they convege to the minimum variance filter. The third shows that the performance of even an NFRN is indistinguishable from that of the Kalman filter for a linear system. In the last example, neither the signals nor the measurements can be described by (1.1) and (1.2). Nevertheless, the neural filters of both types work satisfactorily.

In describing the signal/sensor systems in the following examples, the symbols $w(t)$ and $v(t)$ denote statistically independent standard white Gaussian sequence with mean $E[w(t)] = E[v(t)] = 0$ and variance $E[w(t)^2] = E[v(t)^2] = 1$. The training data for each example are obtained by simulating the system and consist of batches of 200 realizations of 100 consecutive measurements and signals. A batch is used in a training session of 100 epochs (or sweeps). If the convergence of the weights is not reached, a different batch is used in another training session. Continuing in this manner, we stop the training whenever the convergence is reached, i.e. the change in the mean square error $C(w)$ is less than $10^{-10}C(w)$.

The training method of first applying TDRBP to evaluate the gradient dC/dw and optimize C by the conjugate gradient method was effective, albeit slow. A 486-based personal computer was used for training. On the average, it took about 16 hours to train up a neural filter on the PC. Local minima of the error function did not pose a serious problem. Only three training sessions resulted in unsatisfactory filters during our entire training experience. Restarting with a new set of random weights always resolved the problem.

After training, 500 Monte Carlo test runs were performed for the EKF, IEKF, NFFN, and/or NFRN that are considered in each example. The RMSE of the estimates $\hat{x}(t,\omega)$ produced by a filter at time t for the 500 Monte Carlo runs $\omega = 1, \cdots, 500$ is defined by

$$[\frac{1}{500}\sum_{\omega=1}^{500}(x(t,\omega) - \hat{x}(t,\omega))^2]^{1/2}.$$

Therefore the RMSE for a filter is a function of time. The RMSE of each filter considered versus time is plotted for 120 time points. In the last example, the true signal and the estimates produced by NFFN and NFRN for a single run are also plotted versus time. The last 20 time points are included to demonstrate the generalization ability of neural filters.

In all the examples, both the NFFN and NFRN have 7 hidden neurons. Notice that while there are 71 weights in the NFFN, there are only 36 weights in the NFRN.

Example 1. The signal/sensor system is

$$\begin{aligned} x(t+1) &= 1.1\exp(-2x^2(t)) - 1 + 0.5w(t), \\ y(t) &= x^3(t) + 0.1v(t), \end{aligned}$$

where $x(0)$ is Gaussian with mean -0.5 and variance 0.1^2. Note that $x(t+1) = 1.1\ \exp(-2x^2(t)) - 1$ has a global attractor at $x(t) = 0.0844$.

The EKF fails badly. The RMSEs versus time for the IEKF, NFFN, and NFRN are shown in Fig. 2.4. They are plotted by different lines as specified in the caption of the figure. We note that the RMSEs for the NFFN and NFRN are virtually the same. In fact the RMSEs of the NFRN are only worse than those of the NFFN by 0.1% over the 120 time points. The RMSEs for NFFNs with 2, 4, and 8 neurons are plotted versustime in Fig. 2.5. We notice that they converge rapidly.

Example 2. The signal/sensor system is

$$\begin{aligned} x(t+1) &= 1.7\exp(-2x^2(t)) - 1 + 0.1w(t), \\ y(t) &= x^3(t) + 0.1v(t), \end{aligned}$$

where $x(0)$ is Gaussian with mean zero and variance 0.5^2. Note that $x(t+1) = 1.7\ \exp(-2x^2(t)) - 1$ is a chaotic process.

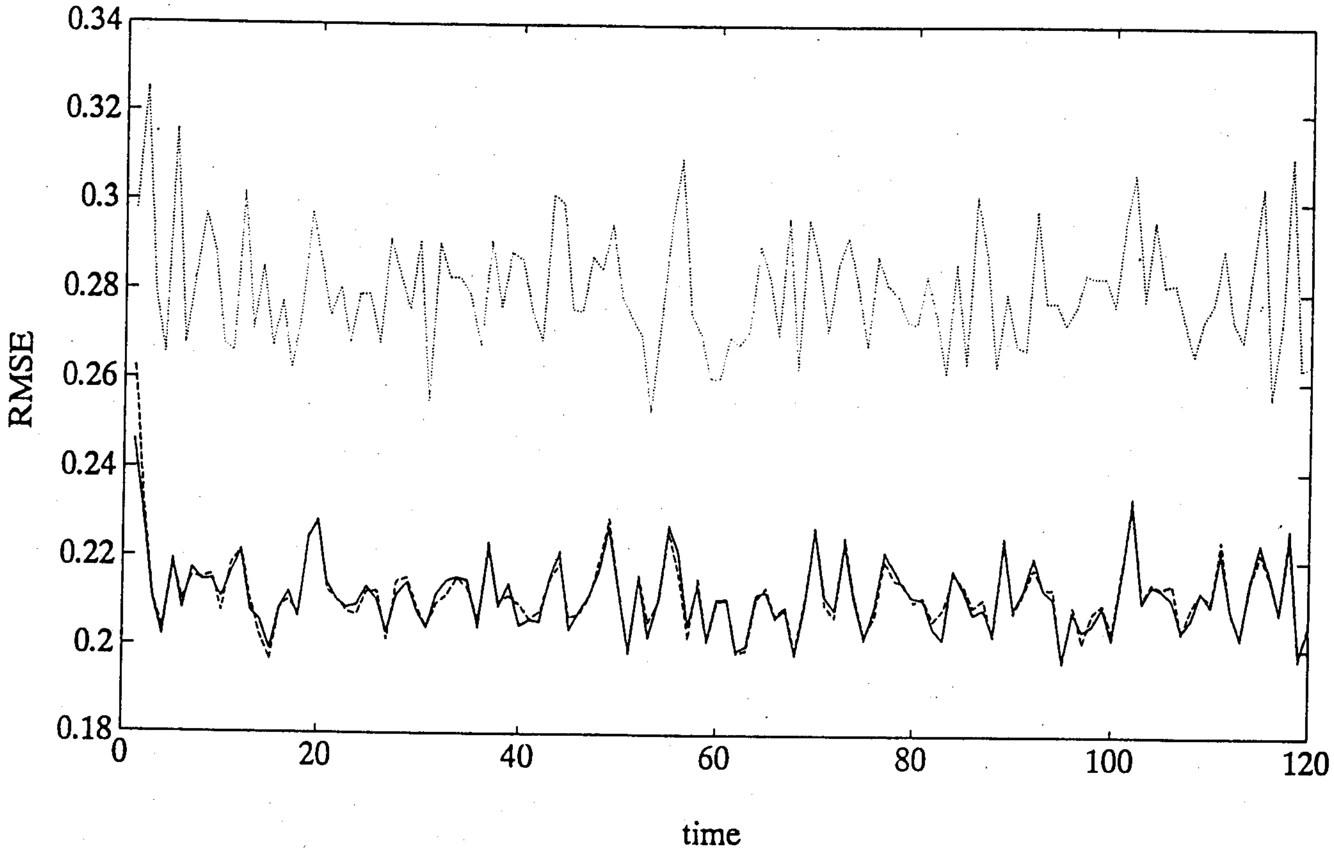


Figure 2.4: RMSEs versus time of IEKF (···), NFFN (—) and NFRN (---) for Example 1. The RMSEs of them over 120 time points are 0.2806, 0.2120 and 0.2122, respectively.

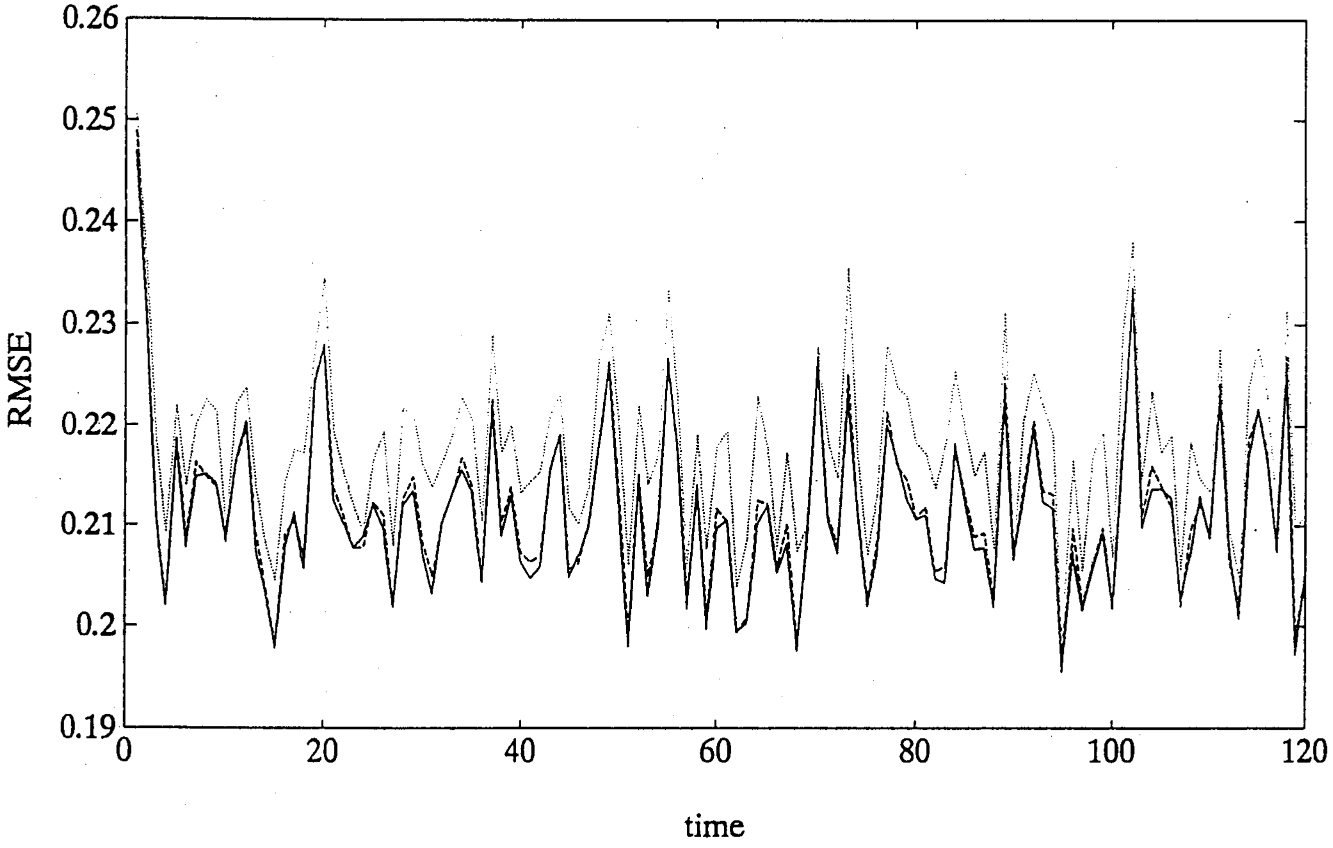


Figure 2.5: RMSEs versus time of NFFNs with 2 neurons (···), 4 neurons (---), and 8 neurons (—) for Example 1. The RMSEs of them over 120 time points are 0.2181, 0.2121 and 0.2114, respectively.

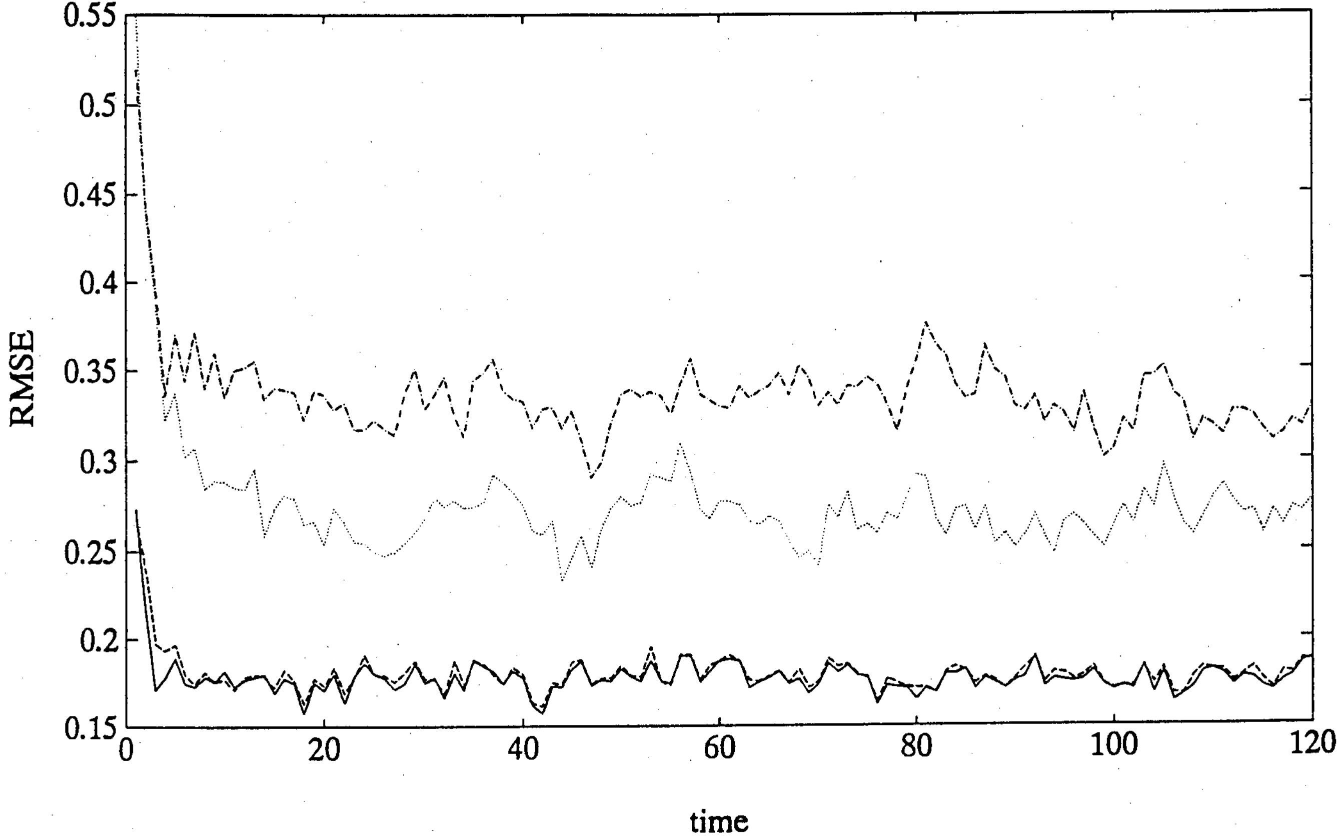


Figure 2.6: RMSEs versus time of EKF (-·-·), IEKF (· · ·), NFFN (—), and NFRN (---) for Example 2. The RMSEs of them over 120 time points are 0.3373, 0.2778, 0.1779 and 0.1804, respectively.

The RMSEs versus time for the EKF, IEKF, NFFN and NFRN are shown in Fig. 2.6. We note that the RMSEs of the NFRN are slightly worse than those of the NFFN by 1.42%. The RMSEs for NFFNs with 2, 4 and 8 neurons are plotted versus time in Fig. 2.7. Again they converge rapidly.

Example 3. The signal/sensor system is

$$\begin{aligned} x(t+1) &= 0.9x(t) + 0.2w(t), \\ y(t) &= x(t) + v(t), \end{aligned}$$

where $x(0)$ is Gaussian with mean zero and variance 1^2.

The RMSEs versus time for the Kalman filter and NFRN are shown in Fig. 2.8. We note that the two lines are virtually the same.

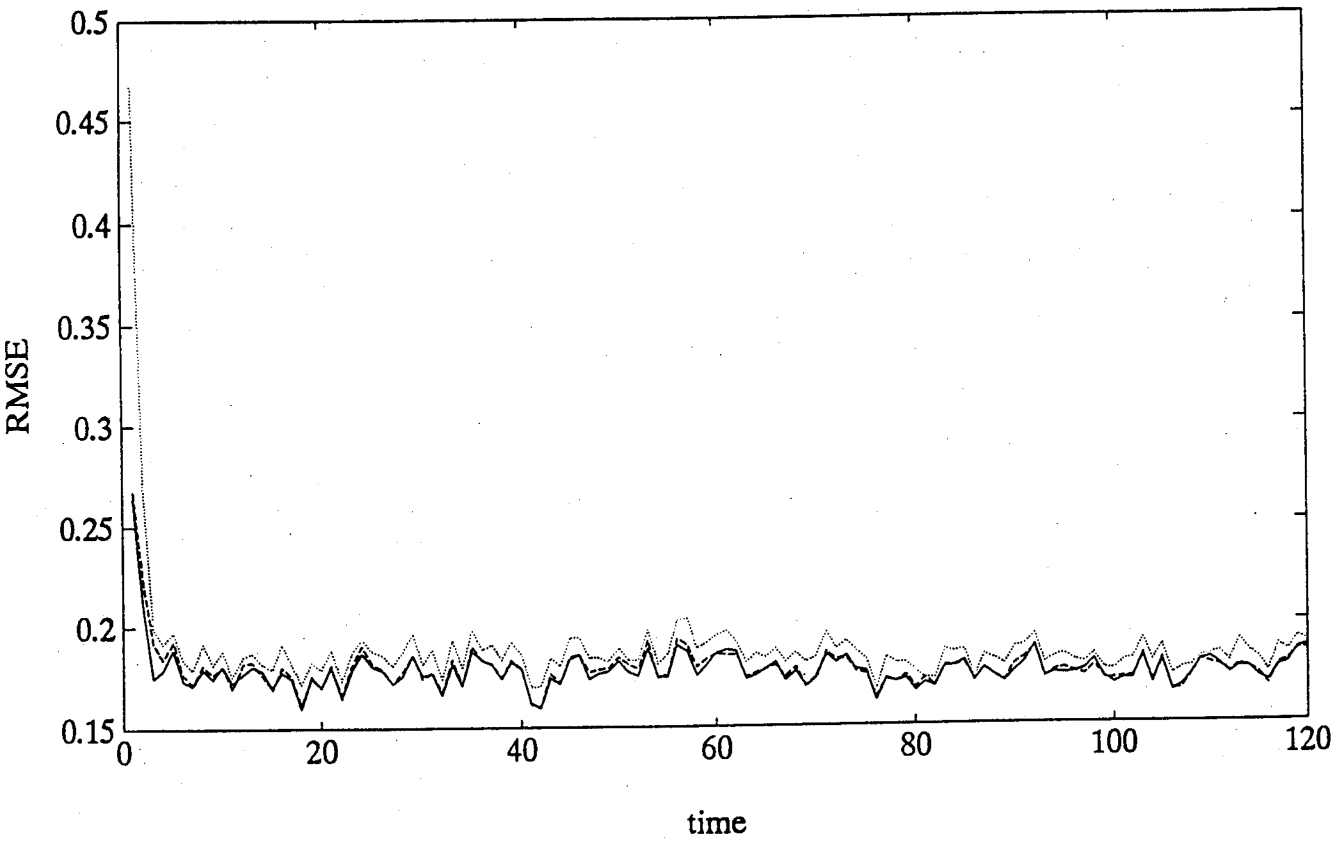


Figure 2.7: RMSEs versus time of NFFNs with 2 neurons (···), 4 neurons (---), and 8 neurons (—) for Example 2. The RMSEs of them over 120 time points are 0.1897, 0.1788 and 0.1777, respectively.

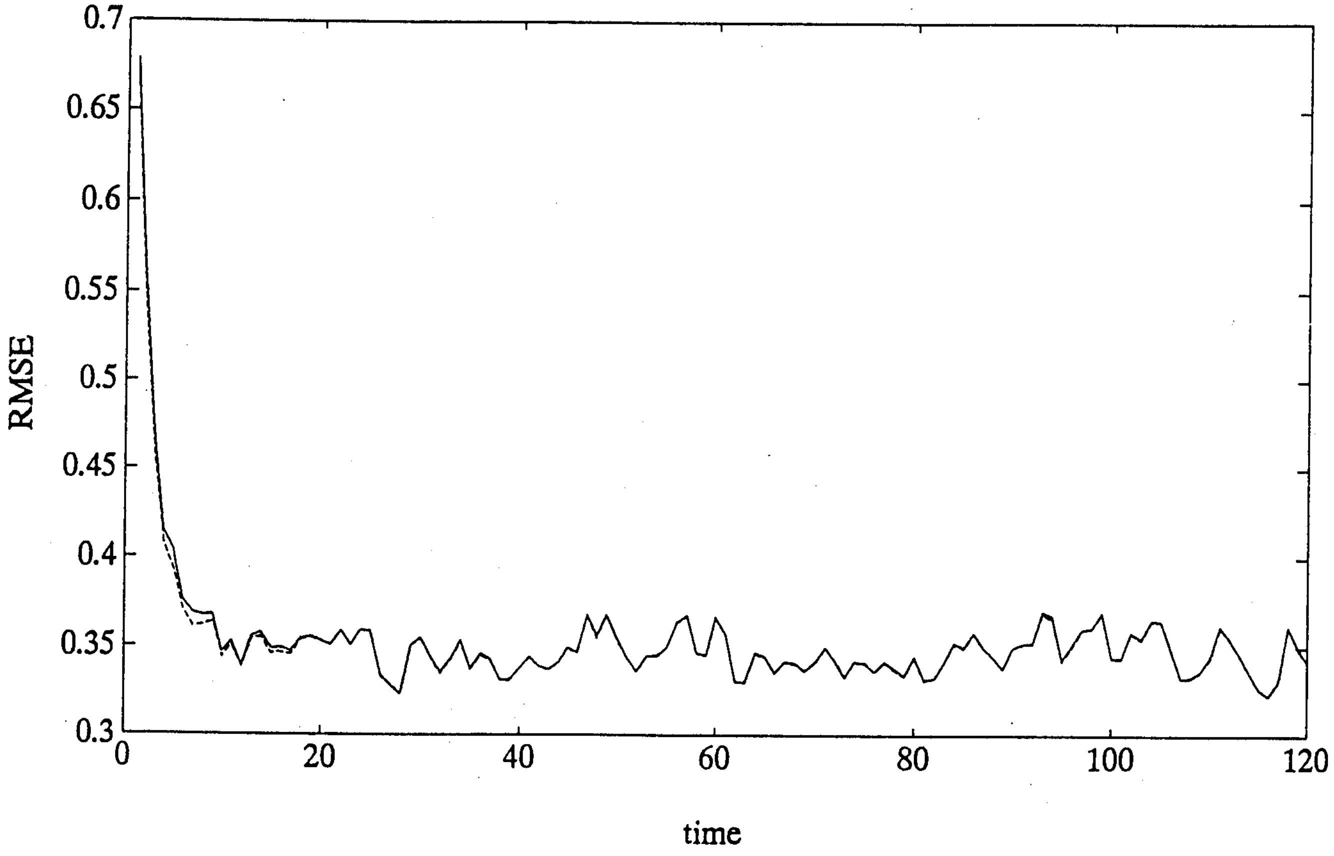


Figure 2.8: RMSEs versus time of the Kalman filter (---) and NFRN (—) for Example 3. The RMSEs of them over 120 time points are 0.3549 and 0.3563, respectively.

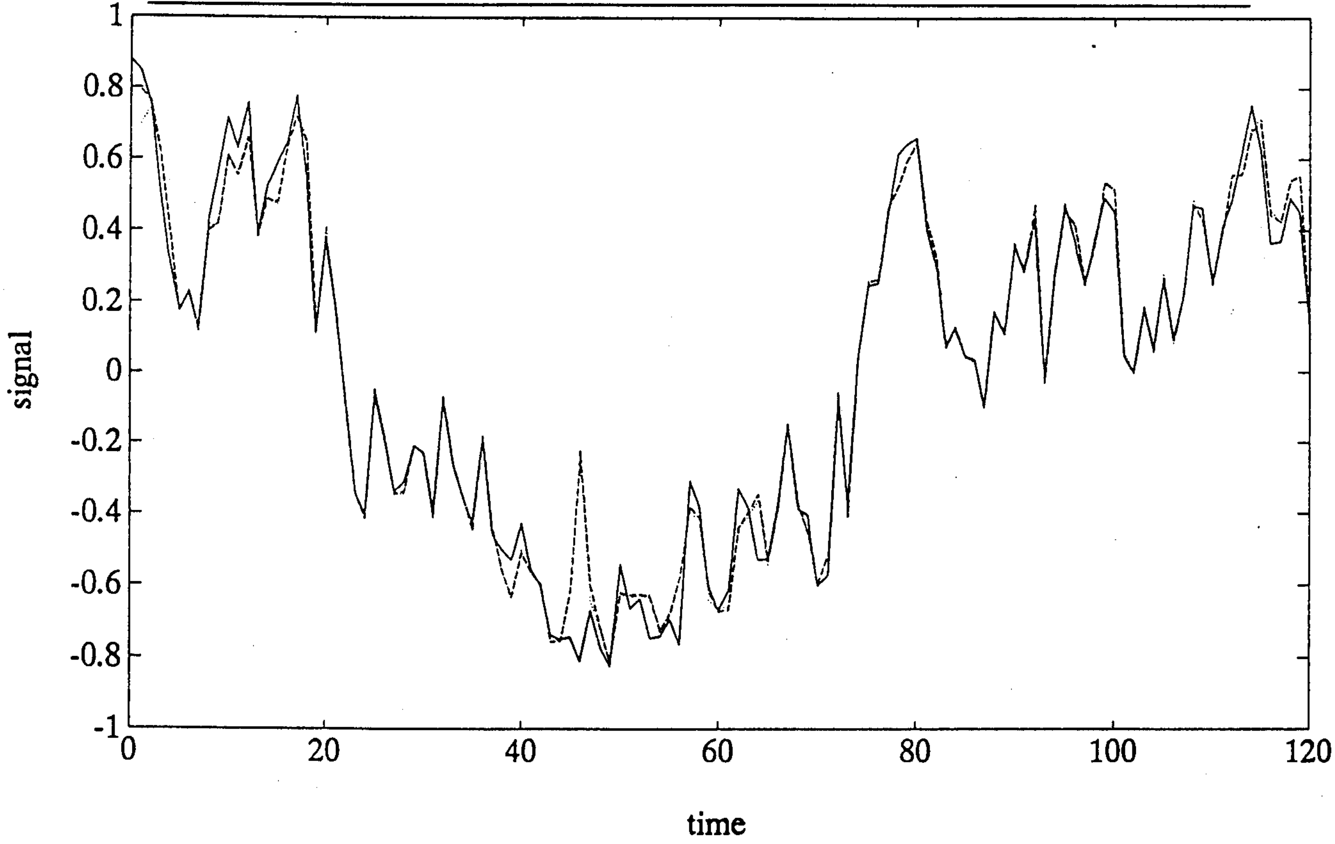


Figure 2.9: The true signal (—) and its estimates produced by NFFN (---) and NFRN (···) versus time for Example 4.

Example 4. The signal/sensor system is

$$\begin{aligned} x(t+1) &= 0.5x(t) + 0.5\tanh(x(t) + 0.5w(t)), \\ y(t) &= x(t) + 0.5x^3(t)v(t), \end{aligned}$$

where $x(0)$ is Gaussian with mean zero and variance 0.5^2. Note that neither the signal nor the measurement process can be transformed into (1.1) or (1.2).

The true signal and its estimates produced by the NFFN and NFRN for a single run are shown in Fig. 2.9. The RMSEs versus time for the NFFN and NFRN are plotted in Fig. 2.10. We note that the RMSEs of the NFRN are worse than those of the NFFN by 0.84%.

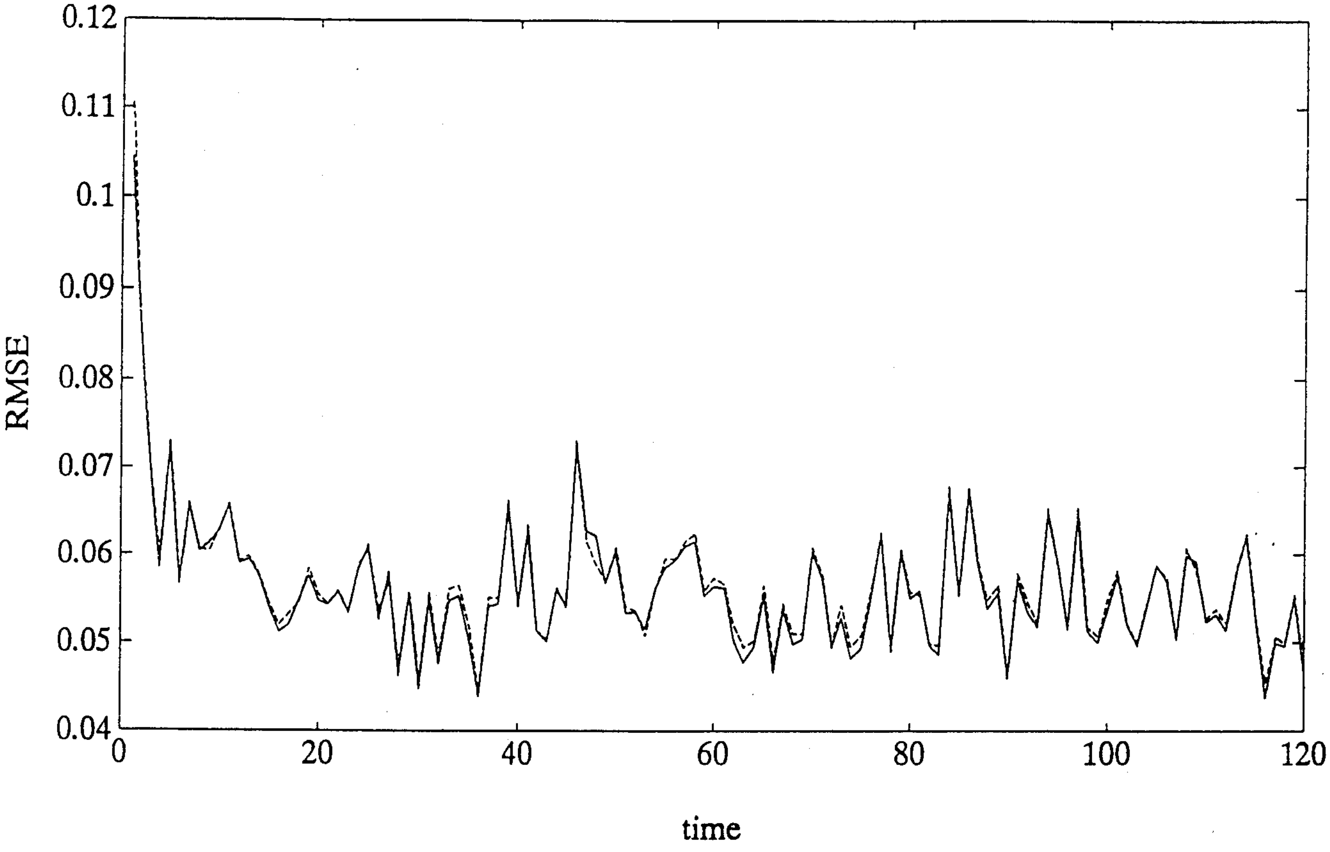


Figure 2.10: RMSEs versus time of NFFN (—) and NFRN (---) for Example 4. The RMSEs of them over 120 time points are 0.0567 and 0.0572, respectively.

3. Optimal Filtering by Multilayer Perceptrons with Output Feedbacks

In this chapter, we will show how an MLP with output feedbacks can be synthesized into a good filter. The synthesis is performed through the neural network training, in which y_t are used as the neural network inputs and x_t as the desired outputs. If a system model such as (1.1) and (1.2) is available, the training data y_t and x_t, are easily generated by computer simulation. Otherwise, the experimental data can be used instead. The training data are all that we need to synthesize the filter.

A requirement here that is not needed in the modern theory of optimal filtering is that x_t and y_t stay in a compact set. The fullfilment of it is easy to justify in the real world. The model, (1.1) and (1.2), violates this requirement. However, it is only an idealization, whose x_t and y_t may stray out of a compact region with arbitrarily small probability provided that the region is sufficiently large. Other than the requirement, no assumption such as the Markov property, the Gaussian distribution, or the linear dynamics is necessary.

The training process minimizes the mean square error between the network outputs $\widehat{\psi}(x_t)$ and the desired outputs $\psi(x_t)$. Consequently, after adequate training, the estimates produced by the network are minimum-variance for the given network architecture. Furthermore, it will be proven that when the size of network increases, the network converges to the minimum variance filter.

There are two types of feedback, namely teacher-forced feedbacks and free feedbacks. From the optimization point of view, teacher forcing amounts to constraints. From the statistical point of view, free feedbacks are allowed to record the most informative statistics at each time instant for the best estimation at the next time instant. Therefore, a free feedback is always better than a teacher-forced feedback, if the application condition allows its use. This point is borne out by our limited simulation results.

Four numerical examples are worked out and reported in this paper. Two of them show that the neural filters consistently outperform the extended Kalman filter and the iterated Kalman filter. The third shows that the performance of the neural filter is virtually indistinguishable from that of the Kalman filter for a linear system of (1.1) and (1.2). In the last example, neither the signal nor the measurement can be described by (1.1) or (1.2). Nevertheless, the neural filter works satisfactorily.

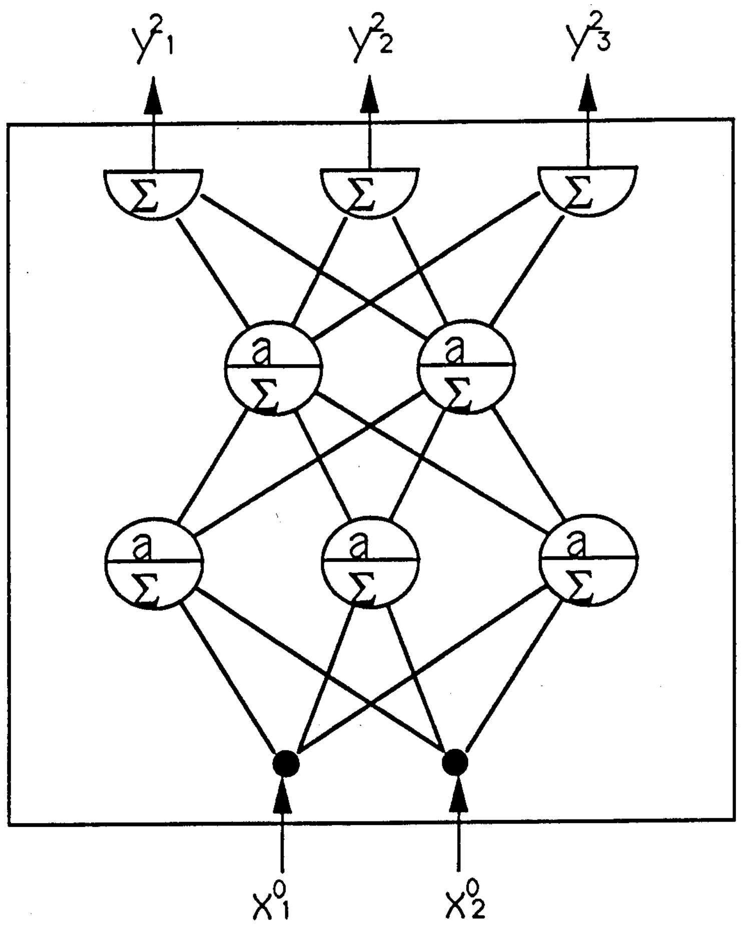


Figure 3.1: A typical multilayer perceptron (MLP).

3.1. Multilayer Perceptrons as Neural Filters

A typical multilayer perceptron (MLP) is illustrated in Fig. 3.1. It has two input nodes, three neurons in the first hidden layer,two neurons in the second hidden layer and three output nodes. Let the weighted sum in the ith neuron in the lth layer and the output of the same be denoted by y^l_i anrd x^l_i, respectively. Thus x^0_i and y^L_i are the ith input and the ith output of an MLP with $L-1$ hidden layers. The outputs of the MLP are obtained by the forward propagation through each

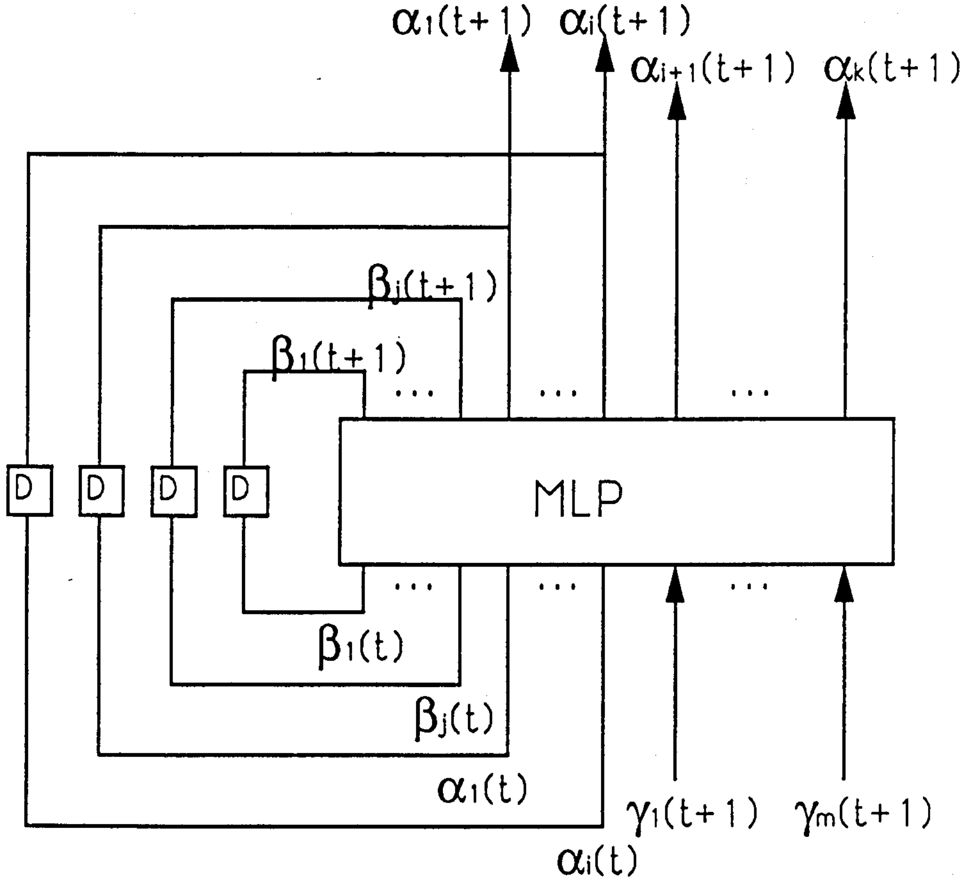


Figure 3.2: The general feedback configuration considered in the chapter.

layer:

$$
\begin{aligned}
y_i^l &= w_{i0}^l + \sum_j w_{ij}^l x_j^{l-1}, \\
x_i^l &= a(y_i^l),
\end{aligned}
$$

for $l = 1, \cdots, L$, where a is the neuron activation function and w_{ij}^l is the weight from the jth neuron in the $(l-1)$th layer to the ith neuron in the lth layer.

The general feedback configuration considered in this paper is shown in Fig. 3.2. It has $j+k$ output nodes, among which $\alpha_1(t+1), \cdots, \alpha_k(t+1)$ are teacher-forced outputs and $\beta_1(t+1), \cdots, \beta_j(t+1)$ are free outputs at time $t+1$. The MLP is trained to have only the teacher-forced outputs mimic the desired outputs. All the free outputs and i of the teacher-forced outputs, say $\alpha_1(t+1), \cdots, \alpha_i(t+1)$, are delayed by one unit of time before being fed back to the input nodes, $\beta_t(t), \cdots, \beta_j(t), \alpha_1(t), \cdots, \alpha_i(t)$. In addition, the MLP has m input nodes on which the external inputs to the MLP, $\gamma_1(t+1), \cdots, \gamma_m(t+1)$, are clamped.

As discussed in the preceding section, by either computer simulation or experiments, we may have available a finite set S of signal/measurement sequences, $(x(t,\omega), y(t,\omega))$, $t = 1, \cdots, T$, $\omega \in S$. It is assumed that the set S is a random sample and adquately reflects the joint probability distributions of the signal and measurement prosesses $x(t)$ and $y(t)$.

Suppose that $\psi(x(t)) = [\psi_1(x(t)), \cdots, \psi_k(x(t))]^T$ are what we want to estimate and are thus the desired outputs for the actual outputs $\alpha_1(t), \cdots, \alpha_k(t)$. Using the m components of the measurement vector $y(t) = [y_1(t), \cdots, y_m(t)]^T$ as the external inputs, $\gamma_1(t), \cdots, \gamma_m(t)$, the training data consists of the I/O pairs $(y(t,\omega), \psi(x(t,\omega)))$, $t = 1, \cdots, T$, $\omega \in S$.

The training is to minimize, by the variation of the MLP weights w, the mean square error

$$C(w) = \frac{1}{T(\#S)} \sum_{\omega \in S} \sum_{t=1}^{T} \sum_{l=1}^{k} (\alpha_l(t,\omega) - \psi_l(x(t,\omega)))^2 \tag{3.1}$$

where $\#S$ is the number of elements in S and $\alpha_l(t,\omega)$ is the actual MLP output corresponding to the input sequence $y(\tau,\omega)$, $\tau = 1, \cdots, t$, at time t. We note here that $\alpha_l(t,\omega)$ is a function of $y(\tau,\omega)$, $\tau = 1, \cdots, t$.

A good training method, which is used in our simulation, is first to evaluate the gradient of the above error function with respect to w by the time-dependent recurrent backpropagation by Werbos[27] and Pearlmutter[20] and then to use it in a conjugate gradient algorithm for optimization.

An adequately trained network through minimizing $C(w)$ will be called a neural filter.

3.2. Convergence to the Minimum Variance Filter

The collection of all k-dimensional random vectors $x = [x_1, \cdots, x_k]^T$ is a Hilbert space L_2 with the norm $E[\| x \|^2] = E[\sum_{l=1}^{k} x_l^2]$, where E denotes the expectation or the average over the sample space Ω with the σ-field $\mathcal{A}$ of events and the probability measure P.

By simple calculation, we have for any Borel-measurable function f of $y^t :=$

$\{y(1), \cdots, y(t)\}$,

$$\begin{aligned} C' &:= \frac{1}{T}\sum_{t=1}^{T} E[\| \psi(x(t)) - f(y^t) \|^2] \\ &= \frac{1}{T}\sum_{t=1}^{T}\{E[\| \psi(x(t)) - E[\psi(x(t))|y^t] \|^2] \\ &\quad + E[\| E[\psi(x(t))|y^t] - f(y^t) \|^2]\}, \end{aligned} \qquad (3.2)$$

which shows that the conditional expectation $E[\psi(x(t))|y^t]$ is the minimum variance estimate of $\psi(x(t))$ given y^t.

Under the assumption that averaging over S is indistinguishable from taking expectation over Ω, we see, by comparing $C(\omega)$ in (3.1) and C' in (3.2), that

$$\begin{aligned} C(w) &= \frac{1}{T(\#S)}\sum_{\omega\in S}\sum_{t=1}^{T} \| \psi(x(t,\omega)) - E[\psi(x(t))|y^t(\omega)] \|^2 \\ &\quad + \frac{1}{T(\#S)}\sum_{\omega\in S}\sum_{t=1}^{T} \| \alpha(t,\omega) - E[\psi(x(t))|y^t(\omega)] \|^2 \end{aligned}$$

Training amounts to minimizing the second term above. The objective is, of course, to minimize $(1/T)\sum_{t=1}^{T} E[\| \alpha(t) - \psi(x(t)) \|^2]$.

Recall the neural filter architecture described in Sec. 2. We observe that adding a neuron to a hidden layer decreases $\min_w C(w)$ and so does adding a feedback, free or teacher-forced. We are now ready to state and prove the main theorem of this paper.

Theorem. Consider the n-dimensional random process $x(t)$ and the m-dimensional random process $y(t)$, $t = 1, \cdots, T$ defined on a probability space $(\Omega, \mathcal{A}, P)$. Assume that the range $\Lambda := \{y(t,\omega)|t = 1, \cdots, T,\ \omega \in \Omega\} \subset R^m$ is compact and ψ is an arbitrary k-dimensional Borel function of $x(t)$ such that the second moments $E[\| \psi(x(t)) \|^2]$, $t = 1, \cdots, T$, are finite. Let $\alpha(t)$ denote the k-dimensional output at time t of the neural filter which has taken the external inputs, $y(1), \cdots, y(t)$, in the given order.

(a). Given $\epsilon > 0$, there exist a neural filter of the architecture described in Sec. 1 such that

$$\frac{1}{T}\sum_{t=1}^{T} E[\| \alpha(t) - E[\psi(x(t))|y^t] \|^2] < \epsilon.$$

(b). If the MLP in the neural filter described in Sec. 2 has a single hidden layer with N neurons, $i = 0$ teacher-forced feedback nodes, and $j = m(T-1)$ free feedback nodes and $\alpha(t)$ is written as $\alpha(t; N)$ here to indicat its dependency on N, then

$$r(N) := \min_w \frac{1}{T} \sum_{t=1}^{T} E[\| \alpha(t, N) - E[\psi(x(t))|y^t] \|^2] \tag{3.3}$$

is monotone decreasing and converges to 0 as N approaches infinity.

Proof. Since (a) is an immediate consequence of (b), we shall only prove (b). It is obvious that $r(N)$, $N = 1, 2, \cdots$, are all bounded from below by 0, the sequence $r(N)$ converges and we denote the limit by $r(\infty)$. To prove that $r(\infty) = 0$, it suffices to show that for any $\epsilon > 0$, there is an integer M such that $|r(M)| < \epsilon$. This will be shown in the following for the case in which the measurement process $y(t)$ is scalar-valued, i.e. $m = 1$. The proof for $m > 1$ is similar and omitted.

As illustrated in Fig. 3.3, the $T-1$ free feedback loops can be so constructed that for $\tau = 1, \cdots, T-1$,

$$\beta_\tau(t+1) = a(\beta_{\tau-1}(t)), \ t = 0, , \cdots, T-1. \tag{3.4}$$

The free feedback output nodes are initialized by $\beta_\tau(0) = 0$, $\tau = 1, \cdots, T-1$. At time $t+1$, the free feedback input nodes are

$$\begin{aligned} &(\beta_{T-1}(t), \cdots, \beta_{t+1}(t), \beta_t(t), \beta_{t-1}(t), \cdots, \beta_1(t)) \\ = \ &(0, \cdots, 0, a^{\circ t}(y(1)), a^{\circ(t-1)}(y(2)), \cdots, a(y(t))), \end{aligned}$$

where $a^{\circ t}$ denote the t-fold composition $a \circ a \circ \cdots \circ a$ of a. Including both the feedback input nodes and the external input node, we have the T-dimensional MLP input vector at time $t+1$,

$$z(t+1) = [\beta_{T-1}(t), \cdots, \beta_1(t), y(t+1)]^T.$$

Recall that

$$\zeta(t+1) := E[\psi(x(t+1))|a^{\circ t}(y(1)), a^{\circ(t-1)}(y(2)), \cdots, a(y(t)), y(t+1)]$$

is a Borel measurable function of $a^{\circ t}(y(1)), \cdots, a(y(t))$, and $y(t+1)$. Let us denote it by $\zeta(t+1) = \lambda_{t+1}(a^{\circ t}(y(1)), \cdots, a(y(t)), y(t+1))$. Now let us consider

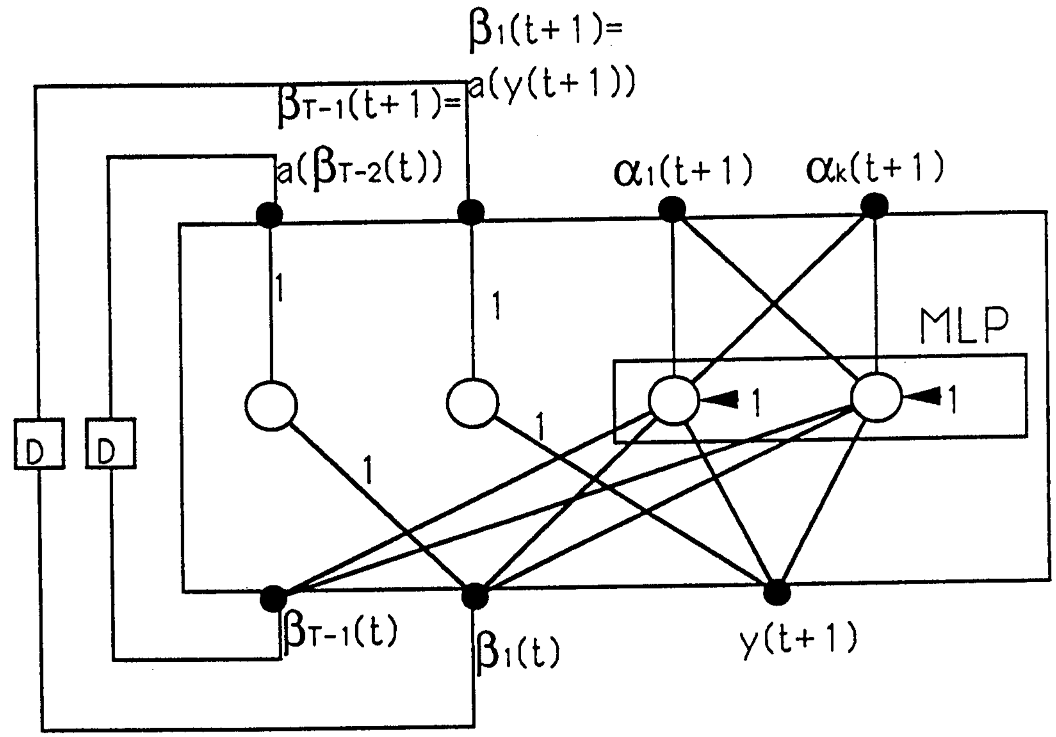


Figure 3.3: A special feedback configuration used in the proof of the main theorem.

the mapping $f : R^T \to R^k : \xi \mapsto \eta = f(\xi)$ defined by

$$\begin{aligned} \eta &= f(\xi_1, \cdots, \xi_T) \\ &= \begin{cases} \lambda_1(\xi_T), & \text{if } \xi_1 = \cdots = \xi_{T-1} = 0, \xi_T \neq 0 \text{ and } \lambda_1(\xi_T) \text{ is defined;} \\ \lambda_2(\xi_{T-1}, \xi_T), & \text{if } \xi_1 = \cdots = \xi_{T-2} = 0, \xi_{T-1} \neq 0, \xi_T \neq 0 \\ & \text{and } \lambda_2(\xi_{T-1}, \xi_T) \text{ is defined;} \\ \vdots & \\ \lambda_T(\xi_1, \cdots, \xi_T), & \text{if } \xi_1 \neq 0, \cdots, \xi_T \neq 0 \text{ and } \lambda_T(\xi_1, \cdots, \xi_T) \text{ is defined;} \\ 0, & \text{otherwise.} \end{cases} \end{aligned}$$

Let us consider also the measures μ_t defined on the Borel sets in R^T by

$$\begin{aligned} \mu_1(d\xi_1 \times d\xi_2 \times \cdots \times d\xi_T) &= \begin{cases} \frac{1}{T} P(\omega | y(1) \in d\xi_T), \\ \qquad \text{if } 0 \in d\xi_t, \text{ for } t = 1, \cdots, T-1, \\ 0, \qquad \text{otherwise;} \end{cases} \\ \mu_2(d\xi_1 \times d\xi_2 \times \cdots \times d\xi_T) &= \begin{cases} \frac{1}{T} P(\omega | a(y(1)) \in d\xi_{T-1}, y(2) \in d\xi_T), \\ \qquad \text{if } 0 \in d\xi_t, \text{ for } t = 1, \cdots, T-2, \\ 0, \qquad \text{otherwise;} \end{cases} \\ &\vdots \\ \mu_T(d\xi_1 \times d\xi_2 \times \cdots \times d\xi_T) &= \frac{1}{T} P(\omega | a^{\circ(T-1)}(y(1)) \in d\xi_1, \cdots, \\ & \quad a(y(T-1)) \in d\xi_{T-1}), y(T) \in d\xi_T). \end{aligned}$$

Notice that $\mu = \mu_1 + \cdots + \mu_T$ is also a mesure on the Borel sets in R^T and that

$$\begin{aligned} & \int \|f(\xi_1, \cdots, \xi_T)\|^2 \mu(d\xi_1 \times d\xi_2 \times \cdots \times d\xi_T) \\ &= \frac{1}{T}\{ \int \|\lambda_1(\xi_T)\|^2 \mu_1(d\xi_1 \times d\xi_2 \times \cdots \times d\xi_T) \\ & \quad + \int \|\lambda_2(\xi_{T-1}, \xi_T)\|^2 \mu_2(d\xi_1 \times d\xi_2 \times \cdots \times d\xi_T) \\ & \quad + \cdots \\ & \quad + \int \|\lambda_T(\xi_1, \cdots, \xi_T)\|^2 \mu_T(d\xi_1 \times d\xi_2 \times \cdots \times d\xi_T)\} \\ &= \frac{1}{T}\{E[\|E[\psi(x(1))|y(1)]\|^2] + E[\|E[\psi(x(2))|a(y(1)), y(2)]\|^2] \\ & \quad + \cdots + E[\|E[\psi(x(T))|a^{\circ(T-1)}(y(1)), a^{\circ(T-2)}(y(2)), \cdots, y(T)]\|^2]\} \end{aligned}$$

$$\begin{aligned}
&\leq \frac{1}{T}\{E[E[\|\psi(x(1))\|^2|y(1)]] + E[E[\|\psi(x(2))\|^2|a(y(1)), y(2)]] \\
&\quad + \cdots + E[E[\|\psi(x(T))\|^2|a^{\circ(T-1)}(y(1)), a^{\circ(T-2)}(y(2)), \cdots, y(T)]]\} \\
&= \frac{1}{T}\{E[\|\psi(x(1))\|^2] + E[\|\psi(x(2))\|^2] + \cdots + E[\|\psi(x(T))\|^2]\} \\
&< \infty.
\end{aligned}$$

Since $y(t)$, $t = 1, \cdots, T$, are assumed to have compact ranges, it is easy to see that $a^{\circ t}(y(\tau))$, t and $\tau = 1, \cdots, T$, all have compact ranges. Hence there is a compact set W in R^T such that $\mu(W) = 1$.

Consider now the normed space $L_2(R^T, \mu)$ with the norm defined by $[\int \|g(\xi_1, \cdots, \xi_T)\|^2\ \mu(d\xi_1 \times \cdots \times d\xi_T)]^{1/2}$ for each k-dimensional function $g \in L_2(R^T, \mu)$. By Corollary 2.2 of the paper [9] by Hornik, Stinchcombe and White, the k-dimensional functions represented by all the MLPs with k output nodes and a single layer of hidden neurons are dense in $L_2(R^T, \mu)$. It then follows that for any $\epsilon > 0$, there is such an MLP that the function $g(\xi_1, \cdots, \xi_T)$ that it represents satisfies

$$\int \|f(\xi_1, \cdots, \xi_T) - g(\xi_1, \cdots, \xi_T)\|^2 \mu(d\xi_1 \times \cdots \times d\xi_T) < \epsilon.$$

Let us translate the left hand side of this inequality from $L_2(R^T, \mu)$ back to the probability space $(\Omega, \mathcal{A}, P)$:

$$\begin{aligned}
&\int \|f(\xi_1, \cdots, \xi_T) - g(\xi_1, \cdots, \xi_T)\|^2 \mu(d\xi_1 \times d\xi_2 \times \cdots \times d\xi_T) \\
&= \frac{1}{T}\{\int \|\lambda_1(\xi_T) - g(0, \cdots, 0, \xi_T)\|^2 \mu_1(d\xi_1 \times d\xi_2 \times \cdots \times d\xi_T) \\
&\quad + \int \|\lambda_2(\xi_{T-1}, \xi_T) - g(0, \cdots, 0, \xi_{T-1}, \xi_T)\|^2 \mu_2(d\xi_1 \times d\xi_2 \times \cdots \times d\xi_T) \\
&\quad + \cdots \\
&\quad + \int \|\lambda_T(\xi_1, \cdots, \xi_T) - g(\xi_1, \cdots, \xi_T)\|^2 \mu_T(d\xi_1 \times d\xi_2 \times \cdots \times d\xi_T)\} \\
&= \frac{1}{T}\{E[\|E[\psi(x(1))|y(1)] - g(0, \cdots, 0, y(1))\|^2] \\
&\quad + E[\|E[\psi(x(2))|a(y(1)), y(2)] - g(0, \cdots, 0, a(y(1)), y(2))\|^2] \\
&\quad + \cdots \\
&\quad + E[\|E[\psi(x(T))|a^{\circ(T-1)}(y(1)), a^{\circ(T-2)}(y(2)), \cdots, y(T)] - g(a^{\circ(T-1)}(y(1)), \\
&\quad a^{\circ(T-2)}(y(2)), \cdots, y(T))\|^2]\}
\end{aligned}$$

$$= \frac{1}{T}\sum_{t=1}^{T} E[\|E[\psi(x(t))|y^t] - g(0,\cdots,0,a^{\circ(t-1)}(y(1)),a^{\circ(t-2)}(y(2)),\cdots,y(t))\|^2]$$

where the last equality holds, because the σ-fields generated by y^t and $\{a^{\circ(t-1)}(y(1))$, $a^{\circ(t-2)}$ $(y(2))$, $\cdots$, $y(t)\}$ are identical due to the monotonocity of a.

We notice that the foregoing approximation network $g(\xi_1,\cdots,\xi_T)$ and the feedback network (3.4) discussed earlier form an MLP with a single hidden layer. The number N of hidden neurons of this MLP is the sum of those of $g(\xi_1,\cdots,\xi_T)$ and of (3.4). Recalling (3.3) we have

$$\begin{aligned} r(N) \;\leq\; & \frac{1}{T}\sum_{t=1}^{T} E[\|E[\psi(x(t))|y^t] - g(0,\cdots,0,a^{\circ(t-1)}(y(1)), \\ & a^{\circ(t-2)}(y(2)),\cdots,y(t))\|^2] < \epsilon, \end{aligned}$$

since $r(N)$ minimizes the error function in (3.3) by the variation of the weights of the MLP of exactly the same architectures as that for g. This completes the proof. □

In real-world application of a filter, the time interval in which the filter is applied is usually very long, i.e. $T \gg 1$. Having $T-1$ feedback loops is impractical. It is perhaps as impractical as having infinitely many neurons. Nevertheless, the above theorem guarantees you may get arbitrarily close to the minimum variance filter, provided that sufficient numbers of feedbacks and neurons are used.

In fact, the number of feedbacks needed depends very much on the properties and the probability distributions of the signal and measurement processes, $x(t)$ and $y(t)$, individually and jointly. After adequate training, the feedbacks actually carry the "most informative" *a priori* statistics at each time instant for producing the "most accurate" *a posteriori* estimates required at the next time instant. For instance, if $x(t)$ and $y(t)$ satisfy (1.1) and (1.2) with f and h being linear in $x(t)$, G constant, and $x(0)$, $v(t)$ and $w(t)$ Gaussian, the feedbacks needed for minimum variance filtering are the conditional mean and covariances of $x(t)$ given y^t, regardless of T.

3.3. Numerical Examples

Four examples are worked out. The first two show that neural filters with various combinations of free and teacher-forced feedbacks outperform the extended

Kalman filter (EKF) and the iterated extended Kalman filter (IEKF) for nonlinear systems described by (1.1) and (1.2). It is also shown that free feedbacks are better than teacher-forced feedbacks and the neural filters with only free feedbacks converge rather fast for the example systems. By the main theorem, they convege to the minimum variance filter. The third shows that the performance of a neural filter is indistinguishable from that of the Kalman filter for a linear system. In the last example, neither the signals nor the measurements can be described by (1.1) and (1.2). Nevertheless, the neural filters work satisfactorily.

In describing the signal/sensor systems in the following examples, the symbols $w(t)$ and $v(t)$ denote statistically independent standard white Gaussian sequences with mean $E[w(t)] = E[v(t)] = 0$ and variance $E[w(t)^2] = E[v(t)^2] = 1$. The training data for each example are obtained by simulating the system and consist of batches of 200 realizations of 100 consecutive measurements and signals. A batch is used in a training session of 100 epochs (or sweeps). If the convergence of the weights is not satisfactory, a different batch is used in another training session. Continuing in this manner, we stop the training whenever the convergence is reached, i.e. the change in the mean square error $C(w)$ is less than $10^{-10}C(w)$.

In training, we first calculate the ordered partial derivatives, $\partial^+ C/\partial y_i(t,\omega)$, $\partial^+ C/\partial \beta_i(t,\omega)$ and $\partial^+ C/\partial \psi_i(x(t,\omega))$, by the time-dependent recurrent backpropagation (TDRBP) derived by Werbos[27] and Pearlmutter[20]. Then the gradient, dC/dw, is evaluated and used in a conjugate gradient method for minimizing $C(w)$.

After training, 500 Monte Carlo test runs were performed for each filter that is considered in each example. The RMSE of each filter considered versus time is plotted for 120 time points. In the last example, the true signal and the estimate produced by the neural filter in a single run are also plotted versus time. The last 20 time points are included to demonstrate the generalization ability of neural filters.

In the following, we shall denote a neural filter with m teacher-forced feedbacks and n free feedbacks by NF(m, n).

Example 1. The signal/sensor system is

$$\begin{aligned} x(t+1) &= 1.1\exp(-2x^2(t)) - 1 + 0.5w(t), \\ y(t) &= x^3(t) + 0.1v(t), \end{aligned}$$

where $x(0)$ is Gaussian with mean -0.5 and variance 0.1^2. Note that $x(t+1) = 1.1\ \exp(-2x^2(t)) - 1$ has a global attractor at $x(t) = 0.0844$.

The EKF fails badly. The RMSEs versus time for the neural filters with 1 feedback, 2 feedbacks and 3 feedbacks are plotted in Figs. 3.4 - 3.7, respectively. All possible combinations of the teacher-forced feedbacks are included for each

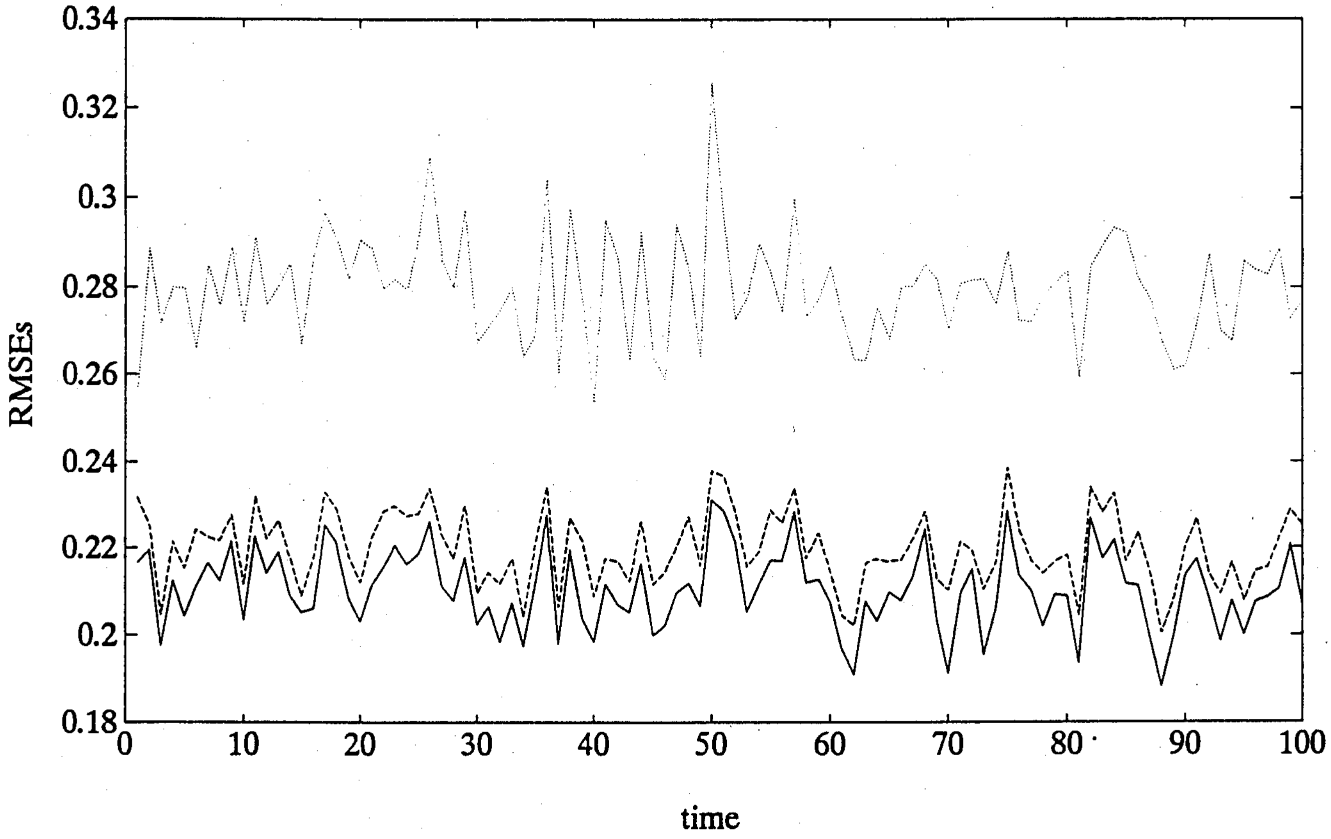


Figure 3.4: RMSEs versus time of IEKF (···), NF(1,0) (---) and NF(0,1) (—) for Example 1. The RMSEs of them over 100 time points are 0.2800, 0.2200 and 0.2106, respectively.

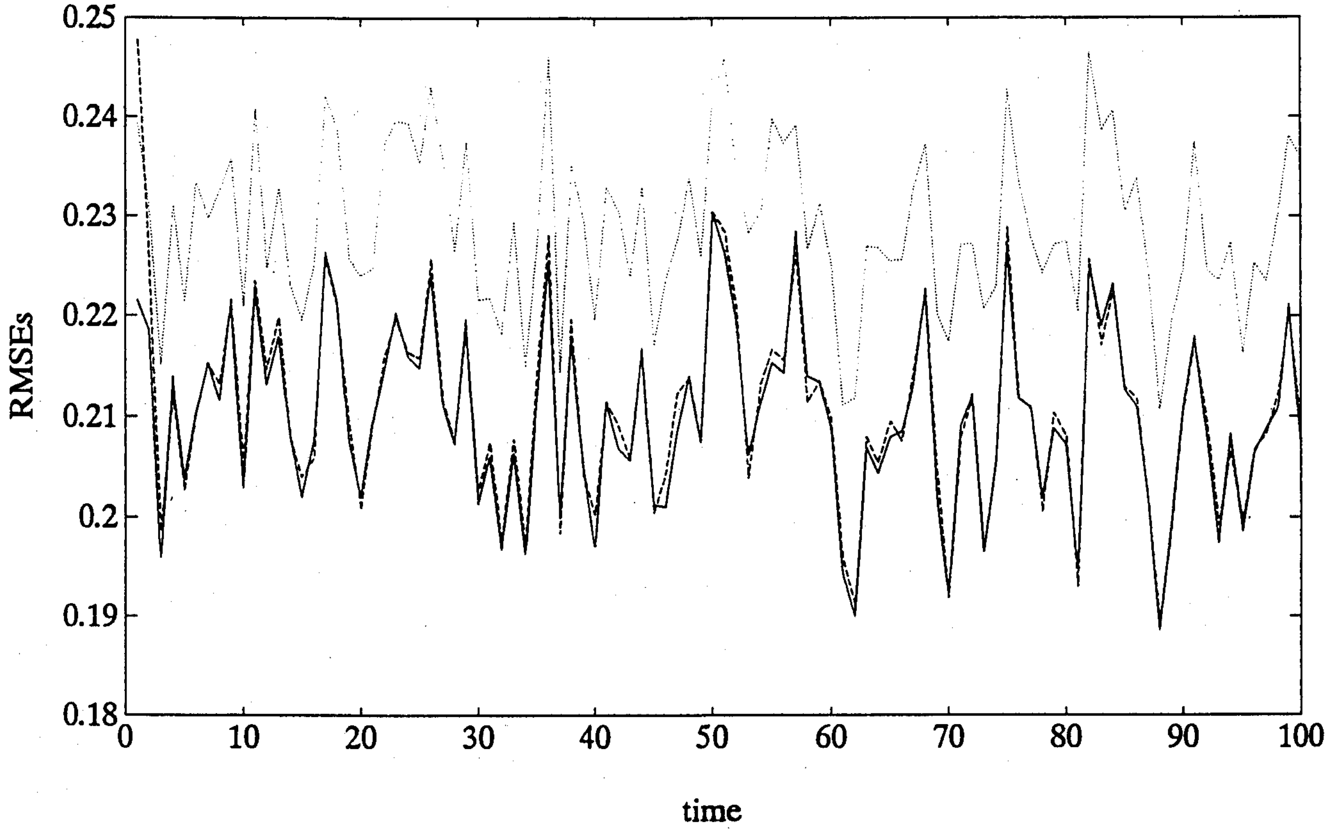


Figure 3.5: RMSEs versus time of NF(2,0) (···), NF(1,1) (---) and NF(0,2) (—) for Example 1. The RMSEs of them over 100 time points are 0.2292, 0.2112 and 0.2103, respectively.

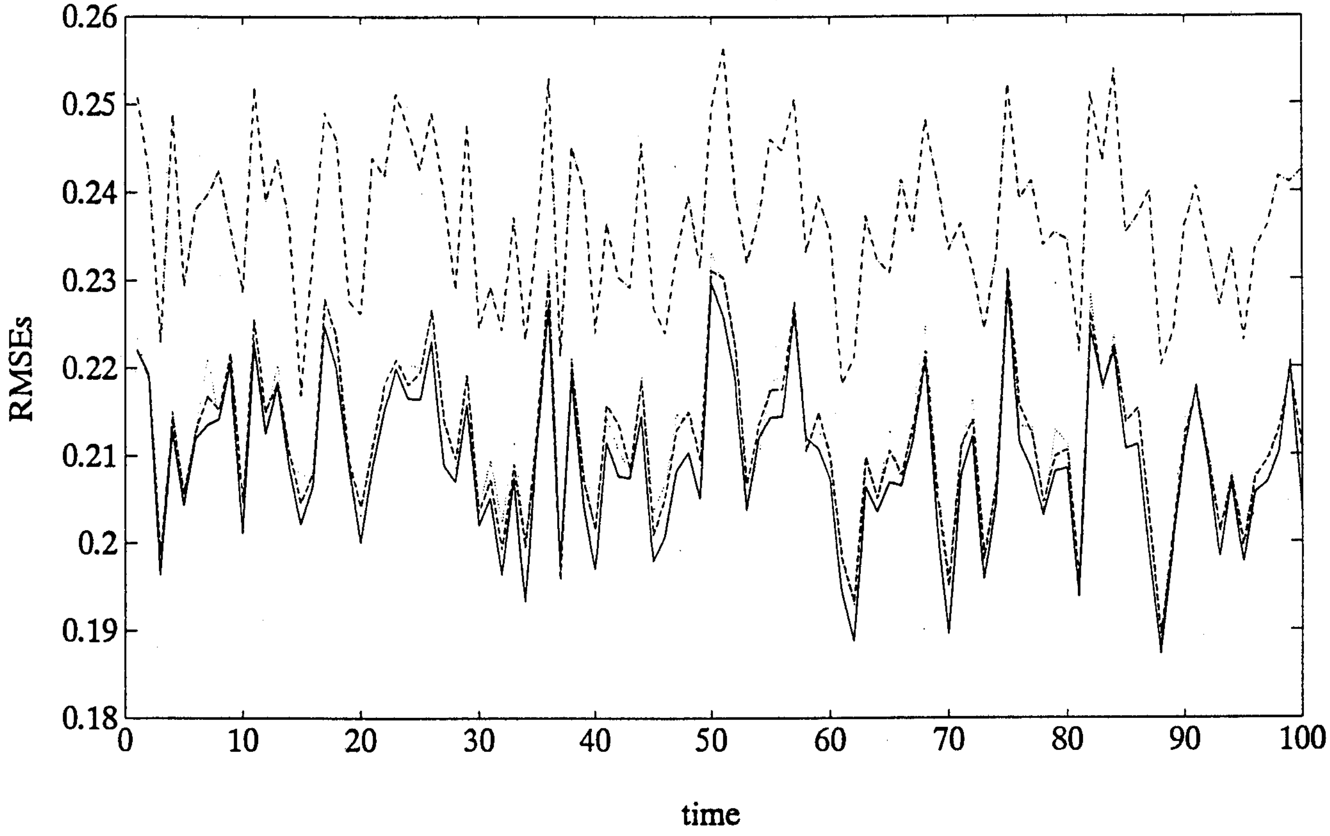


Figure 3.6: RMSEs versus time of NF(3,0) (---), NF(2,1) (· · ·), NF(1,2) (---) and NF(0,3) (—) for Example 1. The RMSEs of them over 100 time points are 0.2369, 0.2124, 0.2121 and 0.2096, respectively.

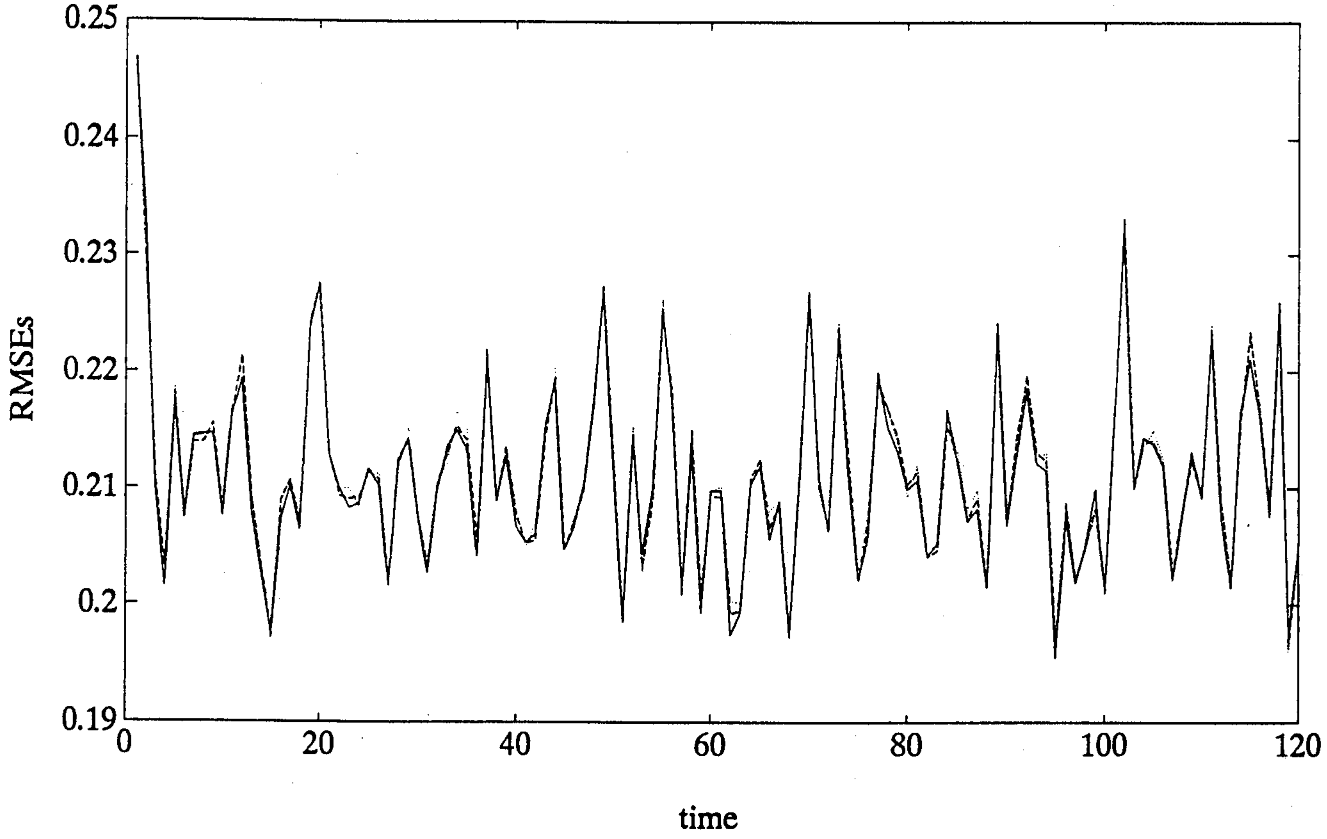


Figure 3.7: RMSEs versus time of NF(0,1) (···), NF(0,2) (---) and NF(0,3) (—) for Example 1. The RMSEs of them over 120 time points are 0.2118, 0.2115 and 0.2113, respectively.

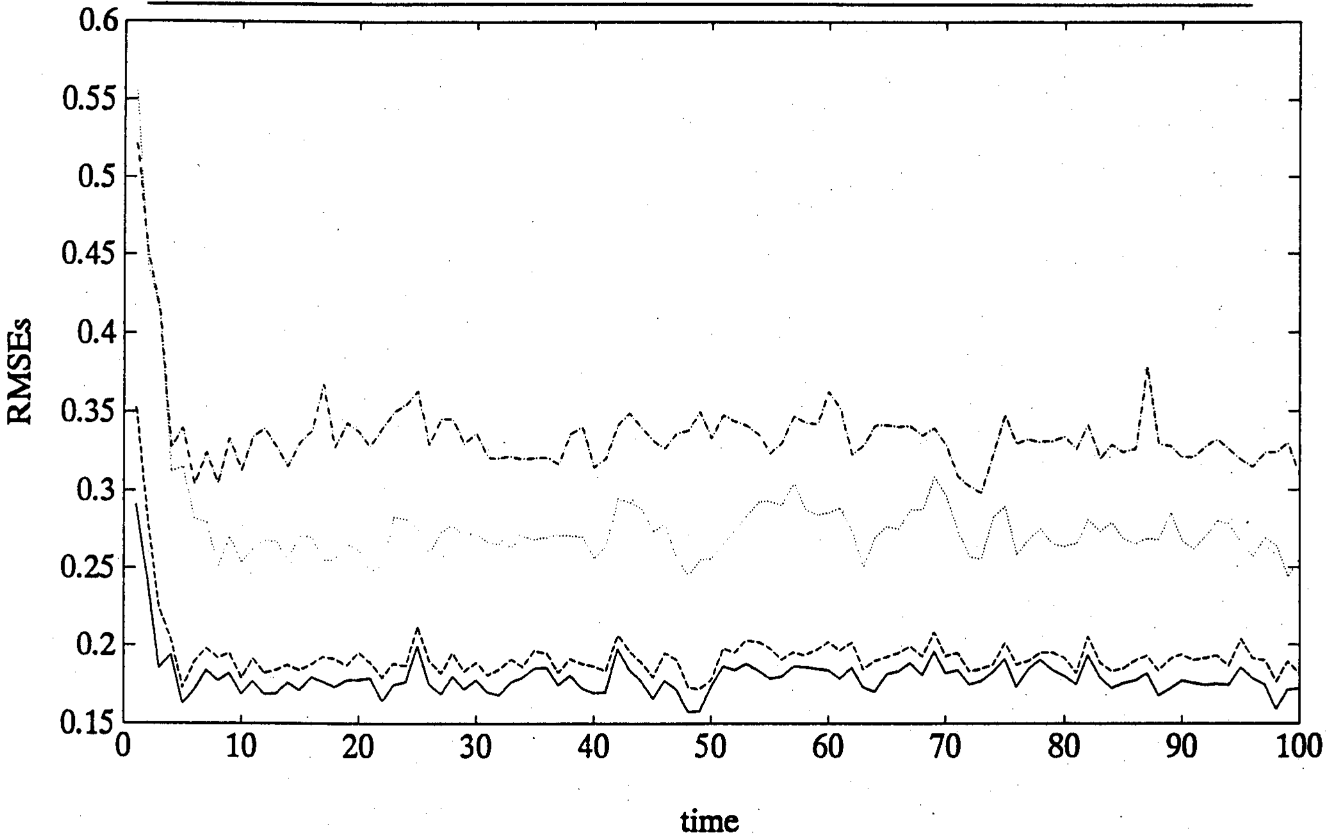


Figure 3.8: RMSEs versus time of EKF (---), IEKF (· · ·), NF(1,0) (---) and NF(0,1) (—) for Example 2. The RMSEs of them over 100 time points are 0.3371, 0.2807, 0.1942 and 0.1801, respectively.

number of feedbacks. The results show that the more free feedbacks there are in the combination, the lower the RMSE is for a fixed total number of feedbacks. Figure 3.7 shows that the performance of the neural filter with only free feedbacks converges as the number of feedbacks increases.

Example 2. The signal/sensor system is

$$\begin{aligned} x(t+1) &= 1.7\exp(-2x^2(t)) - 1 + 0.1w(t), \\ y(t) &= x^3(t) + 0.1v(t), \end{aligned}$$

where $x(0)$ is Gaussian with mean zero and variance 0.5^2. Note that $x(t+1) = 1.7\exp(-2x^2(t)) - 1$ is a chaotic process.

The RMSEs versus time for the neural filters with 1 feedback, 2 feedbacks and 3 feedbacks are plotted in Figs. 3.8 - 3.11, respectively. All possible combinations of the teacher-forced feedbacks are included for each number of feedbacks. The

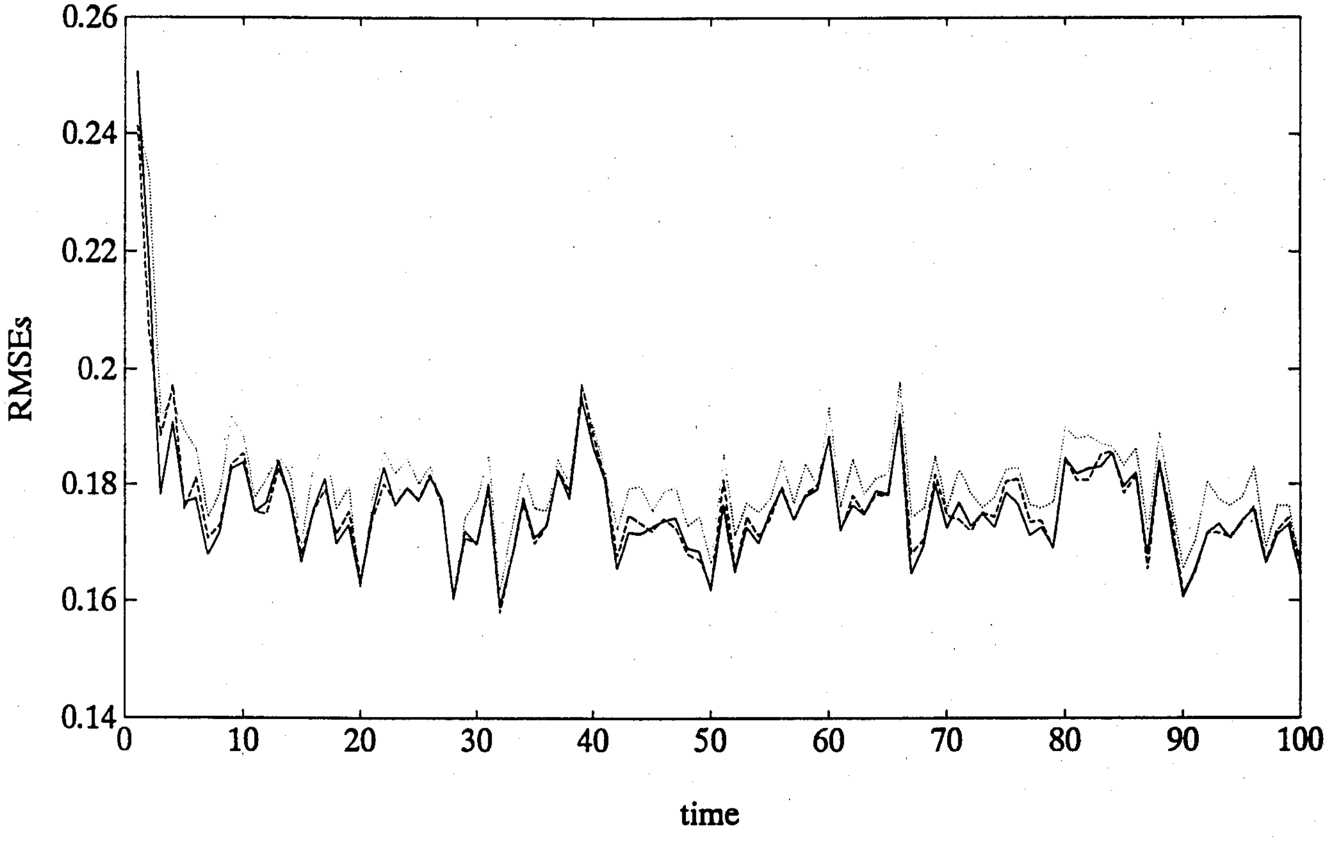


Figure 3.9: RMSEs versus time of NF(2,0) (···), NF(1,1) (---) and NF(0,2) (—) for Example 2. The RMSEs of them over 100 time points are 0.1813, 0.1768 and 0.1766, respectively.

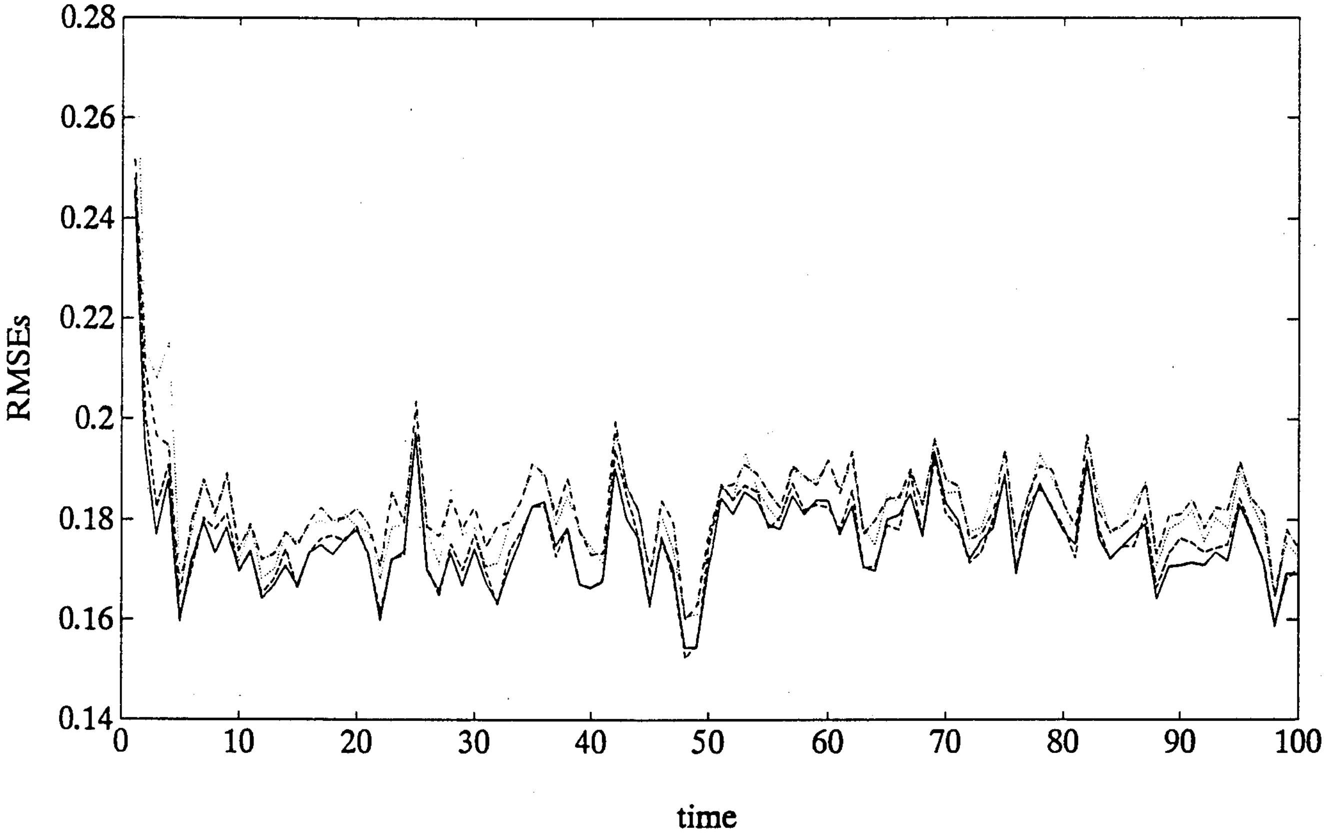


Figure 3.10: RMSEs versus time of NF(3,0) (---), NF(2,1) (· · ·), NF(1,2) (---) and NF(0,3) (—) for Example 2. The RMSEs of them over 100 time points are 0.1836, 0.1833, 0.1774 and 0.1764, respectively.

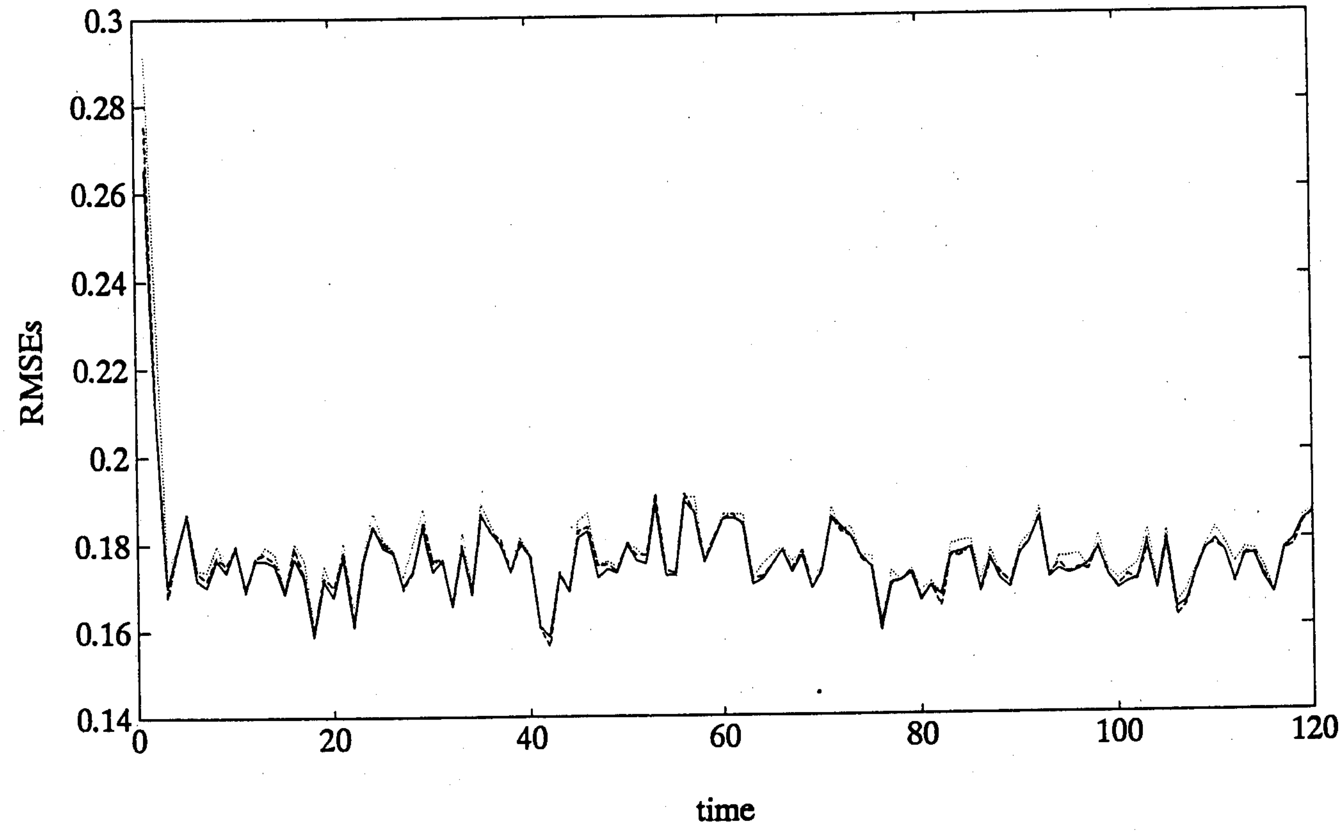


Figure 3.11: RMSEs versus time of NF(0,1) (· · ·), NF(0,2) (---) and NF(0,3) (—) for Example 2. The RMSEs of them over 120 time points are 0.1786, 0.1767 and 0.1763, respectively.

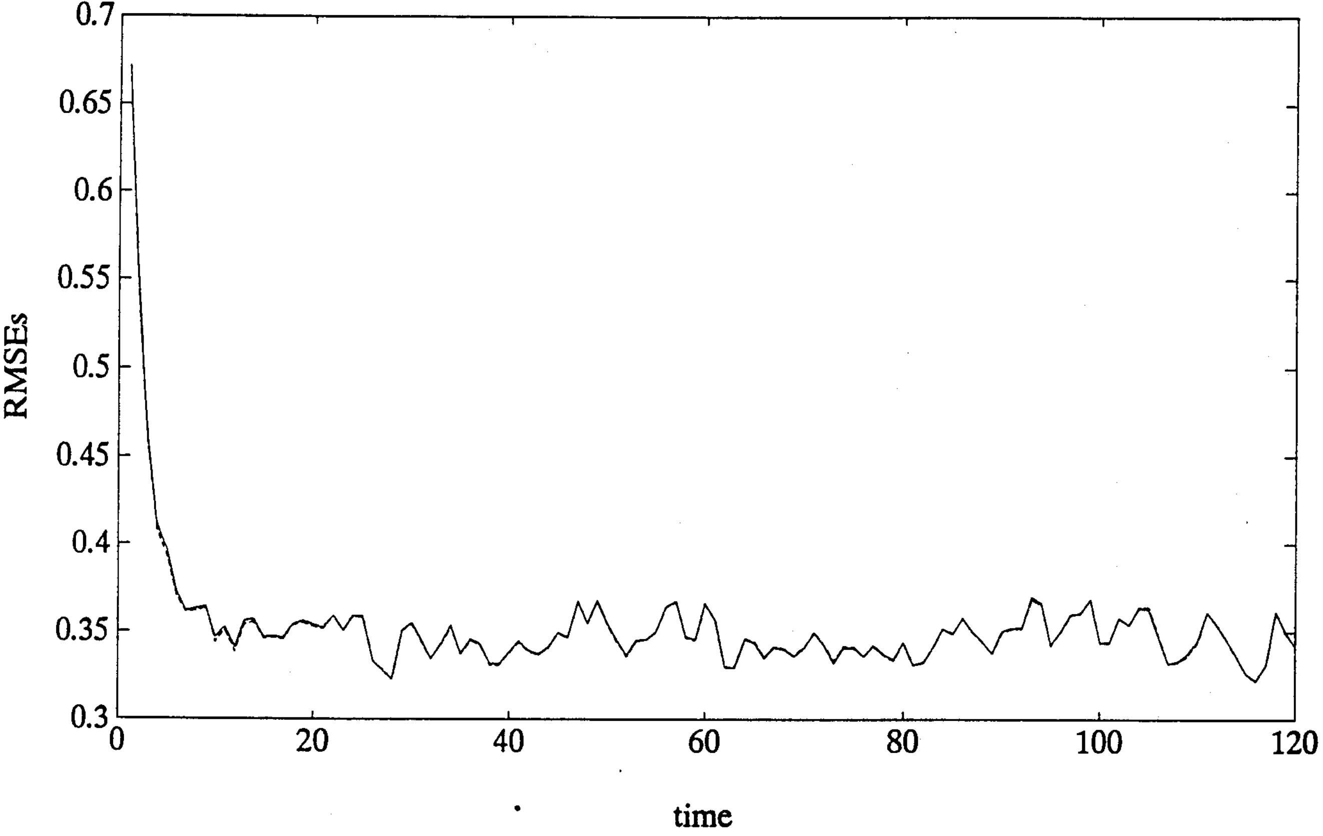


Figure 3.12: RMSEs versus time of the Kalman filter (---) and the NF(0,2) (—) for Example 3. The RMSEs of them over 120 time points are 0.3549 and 0.3555, respectively.

results show that the more free feedbacks there are in the combination, the lower the RMSE is for a fixed total number of feedbacks. Figure 3.11 shows that the performance of the neural filter with only free feedbacks converges as the number of feedbacks increases.

Example 3. The signal/sensor system is

$$\begin{aligned} x(t+1) &= 0.9x(t) + 0.2w(t), \\ y(t) &= x(t) + v(t), \end{aligned}$$

where $x(0)$ is Gaussian with mean zero and variance 1^2.

The RMSEs versus time for the Kalman filter and NF$(0,2)$ are shown in Fig. 3.12. We note that the two lines are virtually the same.

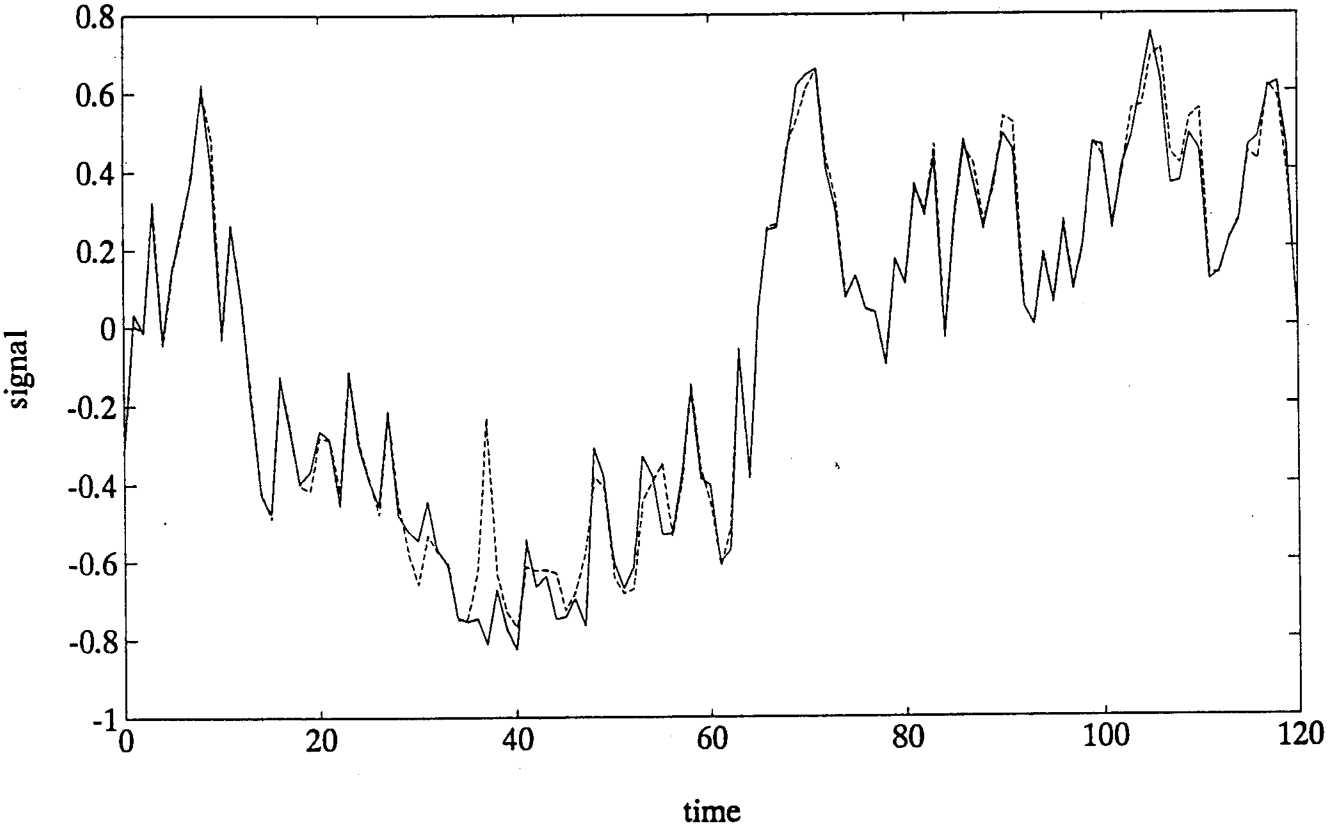


Figure 3.13: The true signal (—) and the estimate (---) produced by the NF(0,3) versus time for Example 4.

Example 4. The signal/sensor system is

$$\begin{aligned} x(t+1) &= 0.5x(t) + 0.5\tanh(x(t) + 0.5w(t)), \\ y(t) &= x(t) + 0.5x^3(t)v(t), \end{aligned}$$

where $x(0)$ is Gaussian with mean zero and variance 0.5^2. Note that neither the signal nor the measurement process can be transformed into (1.1) or (1.2).

The true signal and its estimates produced by the NF$(0,3)$ in a single run are shown in Fig. 3.13. The RMSEs versus time for the NF$(0,1)$, NF$(0,2)$, and NF$(0,3)$ are plotted in Fig. 3.14, which show that the performance of the neural filter with only free feedbacks converges as the number of feedbacks increases.

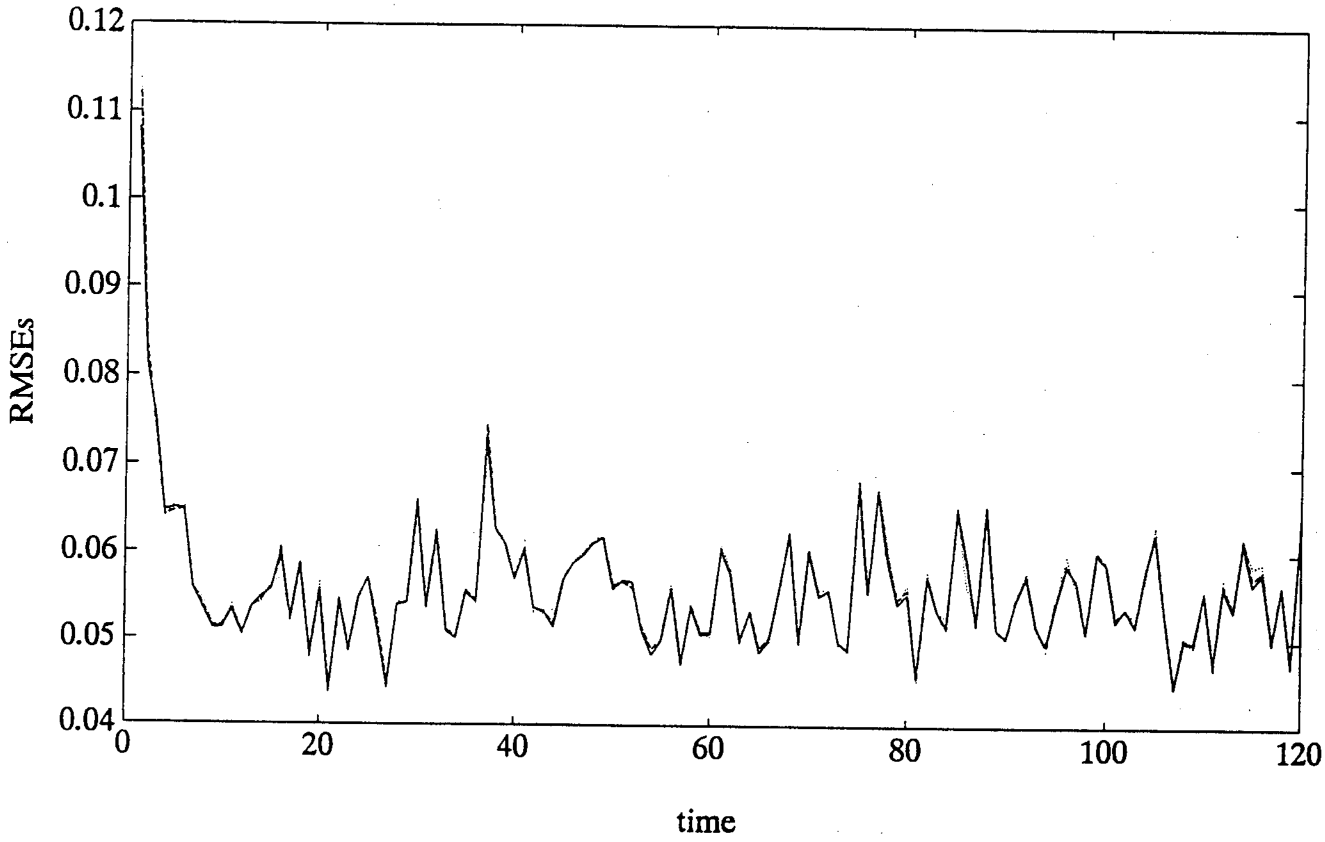


Figure 3.14: RMSEs versus time of NF(0,1) (···), NF(0,2) (---) and NF(0,3) (—) for Example 4. The RMSEs of them over 120 time points are 0.0568, 0.0567 and 0.0565, respectively.

4. Range Extenders and Reducers for Neural Filtering

A fundamental requirement in the neural network approach to optimal filtering is that the measurement process stay in a bounded region. In theory, the requirement is always fulfilled, since no measurable quantities in the real world can not be contained in a bounded region sufficiently large. However, if the measurement process or the signal process or both keep growing such as in a typical filtering problem in satellite orbit determination, aircraft/ship navigation, or target tracking, for a recurrent neural network (RNN) to have a sufficient valid domain to cover the range of measurements and to have a sufficient valid range to cover the range of signals, the sizes of the RNN and the training data set must be large. The larger the RNN and the training data set are, the more difficult it is to train the RNN on the training data set.

Furthermore, the time period or periods, over which the training data is collected, by computer simulation or actual experiment, must be of finite length. If the measurement and signal processes keep growing, the RNN trained on the training data has difficulty to generalize beyond the foregoing time period or periods.

A simple way to extend the RNN output range and to reduce the exogenous input data range is scaling. We may multiply the RNN outputs by a constant greater than one and/or divide the exogenous inputs by another constant also greater than one. Or alternatively, we may use the inverse of a bounded and monotone increasing function to extend (or antisquash) the RNN outputs and/or use another bounded and monotone increasing function to reduce (or squash) the exogenous inputs. However, scaling only suppresses or expands all the "variability." It does not change the "variability." Consequently, its usefulness is limited, as borne out in our numerical experiments.

Our idea for extending the RNN output range is to add some estimate of each desired output to its corresponding RNN output so that the RNN output is intended only to approximate the difference between the desired output and the estimate of it. The estimate does not have to be very good. As long as the range of the differences is an acceptable range for the RNN outputs, the purpose of the estimate is served. A device that generates such an estimate of each desired output, and add it to the corresponding RNN output is called a range extender.

Our idea for reducing the exogenous input data range is to subtract some estimate of each exogenous input from the exogenous input so that the actual input to the RNN is the difference between the exogenous input and its estimate. The estimate does not have to be very good. As long as the range of the differences is an acceptable domain for the RNN, the purpose of the estimate is served. A device that generates such an estimate of each exogenous input and subtracts the estimate from the exogenous input is called a range reducer.

It is necessary to stress here that the foregoing estimate of a desired output and the foregoing estimate of an exogenous input are not arbitrary, they have to be so generated that the system consisting of the RNN together with the range extender and/or range reducer will have a filtering performance surpassing that of the RNN alone. Such a system will be called a neural filter.

We will herein disclose two range extenders and one range reducer. With the aid of one of the two range extenders and/or the range reducer, the output range and/or the input range of an RNN can be maintained so small and the "variability" in the inputs and/or outputs of the RNN can be maintained so low that the neural filter, resulting from training the entire system jointly, outperforms greatly the same RNN aided or not by scaling methods, when the signal and/or measurement processes keep growing in time.

The first range extender herein disclosed uses an extended Kalman filter to provide an estimate of each desired output, that is added to the corresponding RNN output to extend the RNN outputs. Since a standard mathematical/statistical model of the signal and measurement processes is needed for the extended Kalman filter, the first range extender can not be applied where such a mathematical/statistical model is unavailable.

The second range extender herein disclosed can be applied, whether a mathematical/statistical model for the signal and measurement processes is available or not. The second range extender is actually an accumulator that accumulates the outputs of an RNN up to and including the current time point and presents the accumulated value as an estimate of the desired output at the current time point. This amounts to using the preceding estimate presented by the accumulator as an estimate of the current desired output, which is added to the current RNN output to extend the RNN output range.

The range reducer herein disclosed is a differencer that evaluates the difference between a current exogenous input and the preceding exogenous input. Here the preceding exogenous input is looked up as an estimate of the current exogenous input, which is subtracted from the current exogenous input to reduce the range of the exogenous inputs. This range reducer can also be applied whether a mathematical/statistical model is available or not.

4.1. A Range Extender by Kalman Filtering

Assume that the following standard model for the signal process $x(t)$ and measurement process $y(t)$ is given:

$$\begin{aligned} x(t+1) &= f(x(t),t) + G(x(t),t)\xi(t), \ x(1) = x_1, & (4.1) \\ y(t) &= h(x(t),t) + \varepsilon(t), & (4.2) \end{aligned}$$

where $x(t)$ is an n-dimensional stochastic process; $y(t)$ is an m-dimensional stochastic process; x_1 is a Gaussian random vector, $\xi(t)$ and $\varepsilon(t)$ are respectively n_1 dimensional and m_1-dimensional Gaussian noise processes with zero means; x_1, $\xi(t)$ and $\varepsilon(t)$ have given joint probability distributions; and $f(x,t)$, $G(x,t)$ and $h(x,t)$ are known functions with such appropriate dimensions and properties that (4.1) and (4.2) describe faithfully the evolutions of the signal and measurement.

Consider the neural filter depicted in Fig. 4.1. The extended Kalman filter (EKF) and the adder constitute a range extender. The EKF uses the estimate $\hat{x}(t-1)$ from the RNN, the error covariance matrix $\Sigma(t-1)$ from itself, and the measurement vector $y(t)$ to produce an estimate $\hat{x}(t)$ of $x(t)$. The RNN is then employed to only estimate the difference $x(t) - \hat{x}(t)$, whose range is expected to be much smaller than $x(t)$, if $\hat{x}(t)$ is a good estimate of $x(t)$. Denoting the output of the RNN by $\beta^L(t)$, the estimate $\hat{x}(t)$ of $x(t)$ generated by the neural filter is $\hat{x}(t) = \hat{x}(t) + \beta^L(t)$.

The EKF equations are,

$$\begin{aligned} \Sigma(t|t-1) &= F(t-1)\Sigma(t-1)F^T(t-1) + G(\hat{x}(t-1), t-1) \\ &\quad \cdot Q(t-1)G^T(\hat{x}(t-1), t-1) & (4.3) \\ \hat{\hat{x}}(t|t-1) &= f(\hat{x}(t-1), t-1) & (4.4) \\ \Omega(t) &= H^T(t)\Sigma(t|t-1)H(t) + R(t) & (4.5) \\ L(t) &= \Sigma(t|t-1)H(t)\Omega^{-1}(t) & (4.6) \\ \hat{\hat{x}}(t) &= \hat{\hat{x}}(t|t-1) + L(t)[y(t) - h(\hat{\hat{x}}(t|t-1))] & (4.7) \\ \Sigma(t) &= \Sigma(t|t-1) - \Sigma(t|t-1)H(t)\Omega^{-1}(t)H^T(t)\Sigma(t|t-1) & (4.8) \end{aligned}$$

where $F(t-1) = (\partial f(x,t-1)/\partial x)|_{x=\hat{x}(t-1)}$, and $H^T = (\partial h(x,t)/\partial x)|_{x=\hat{\hat{x}}(t|t-1)}$. The training of the RNN has to take into consideration the feedback of $\hat{x}(t-1)$ to EKF. A training algorithm is provided in Sec. V for training a multilayer perceptron with interconnected neurons (MLPWIN) as the RNN in the neural filter. A numerical example using an EKF and an adder as a range extender is given in Sec. IV.

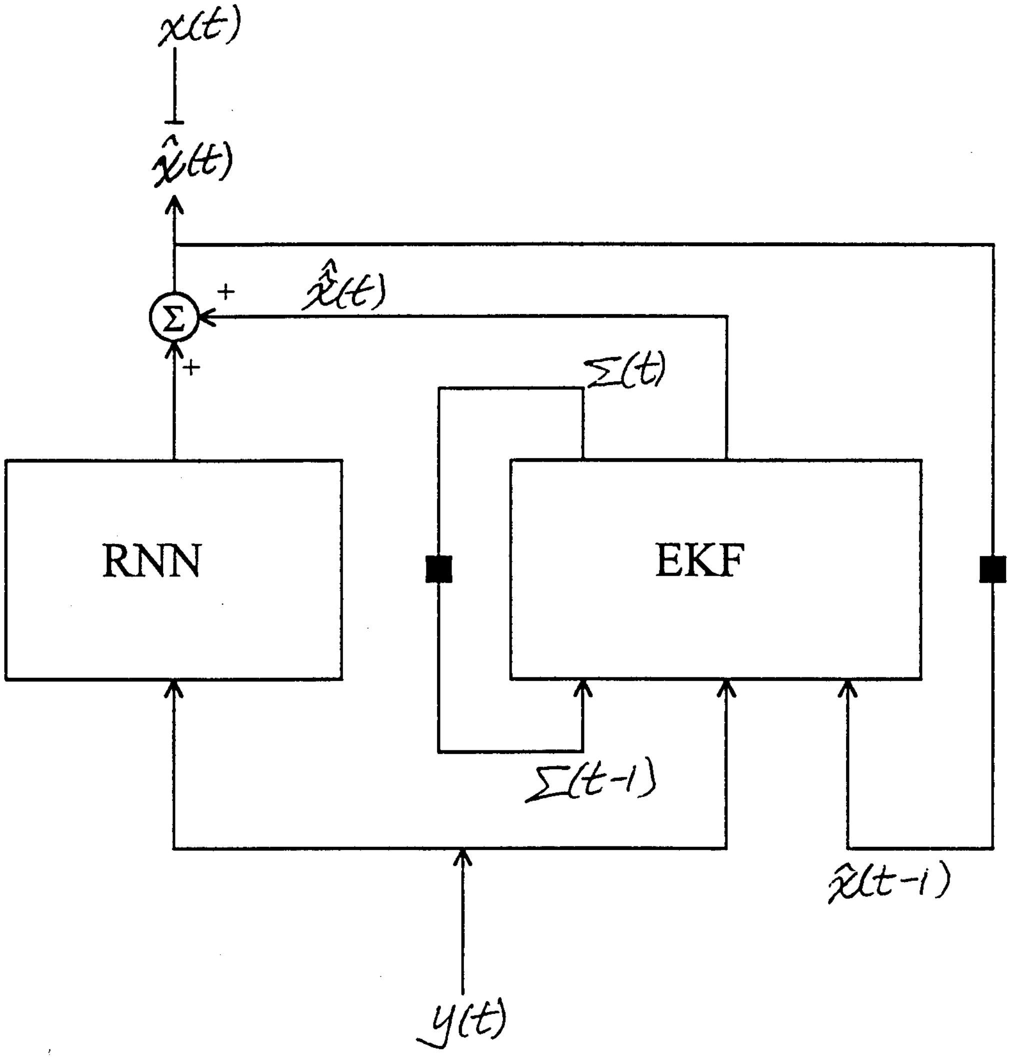


Figure 4.1: Neural filter with an EKF

4.2. An Accumulator as a Range Extender

If the signal process $x(t)$ consists of the vector-values, at discrete time points, of a continuous continuous-time process, then the vector value $x(t-1)$ and a good estimate of it are reasonably good estimates of the vector value $x(t)$. This observation motivated a simple, yet effective way to extend the output range of an RNN in a neural filter, when two consecutive signals, $x(t-1)$ and $x(t)$, are not too far apart.

Consider the neural filter depicted in Fig. 4.2. An accumulator is concatenated at the output terminals of an RNN. At each time point t, the accumulator adds the output vector $\beta^L(t)$ of the RNN to the accumulator's output vector at the preceding time point $t-1$. Thus the accumulator accumulates all the output vectors of the RNN from $t = 1$ onward plus the initial accumulation denoted by $\hat{x}(0)$. Mathematically, the accumulator is described simply by

$$\hat{x}(t) = \beta^L(t) + \hat{x}(t-1), \ t = 1, 2, \cdots, T. \tag{4.9}$$

Here, the RNN actually estimates the difference $x(t) - \hat{x}(t-1)$, which is expected to have a much smaller range than does $x(t)$. If a good a priori estimate $\bar{x}(0)$ is given of $x(0)$, it should be used as the initial accumulation $\hat{x}(0)$. Otherwise, the initial accumulation $\hat{x}(0)$ can be determined together with the weights and/or parameters w and the initial dynamic state v of the RNN in minimizing a training criterion for the neural filter. A training algorithm is provided in Sec. V for training a multilayer perceptron with interconnected neurons (MLPWIN) as the RNN in the neural filter.

Example 1. We tested the aforementioned accumulator using the following signal and measurement model:

$$\begin{aligned} x(t+1) &= x(t) + 1.2 \sin x(t) + 1.21 + 0.2\xi(t), & (4.10) \\ y(t) &= \sin x(t) + 0.1\varepsilon(t), & (4.11) \end{aligned}$$

where $\xi(t)$ and $\varepsilon(t)$ are independent standard white Gaussian sequences, and $x(0) = 0$. Note that even though measurement $y(t)$ is confined essentially in a compact domain, the system remains observable because $y(t)$ is sensitive to the change of $x(t)$ no matter how large $x(t)$ is. So we expect the error of a good estimator to be bounded as $x(t)$ increases.

The filtering result of a neural filter with an accumulator (NFWA) as a range extender was compared with the extended Kalman filter (EKF), the iterated extended Kalman filter (IEKF), and a neural filter (NF) using sigmoidal scaling to

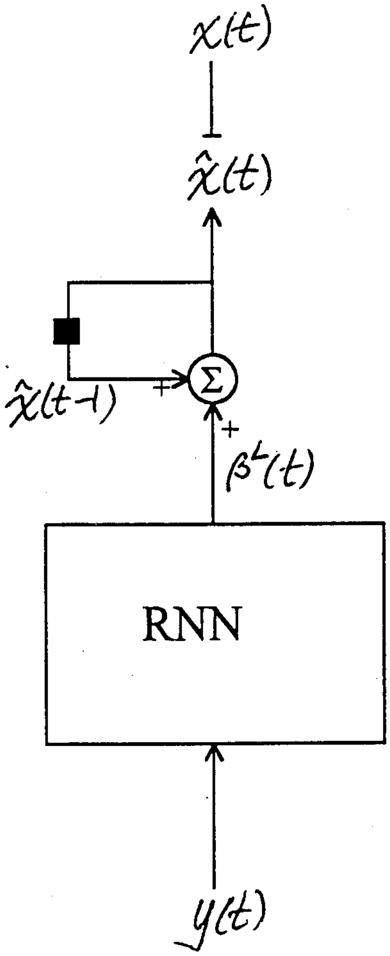


Figure 4.2: Neural filter with an accumulator

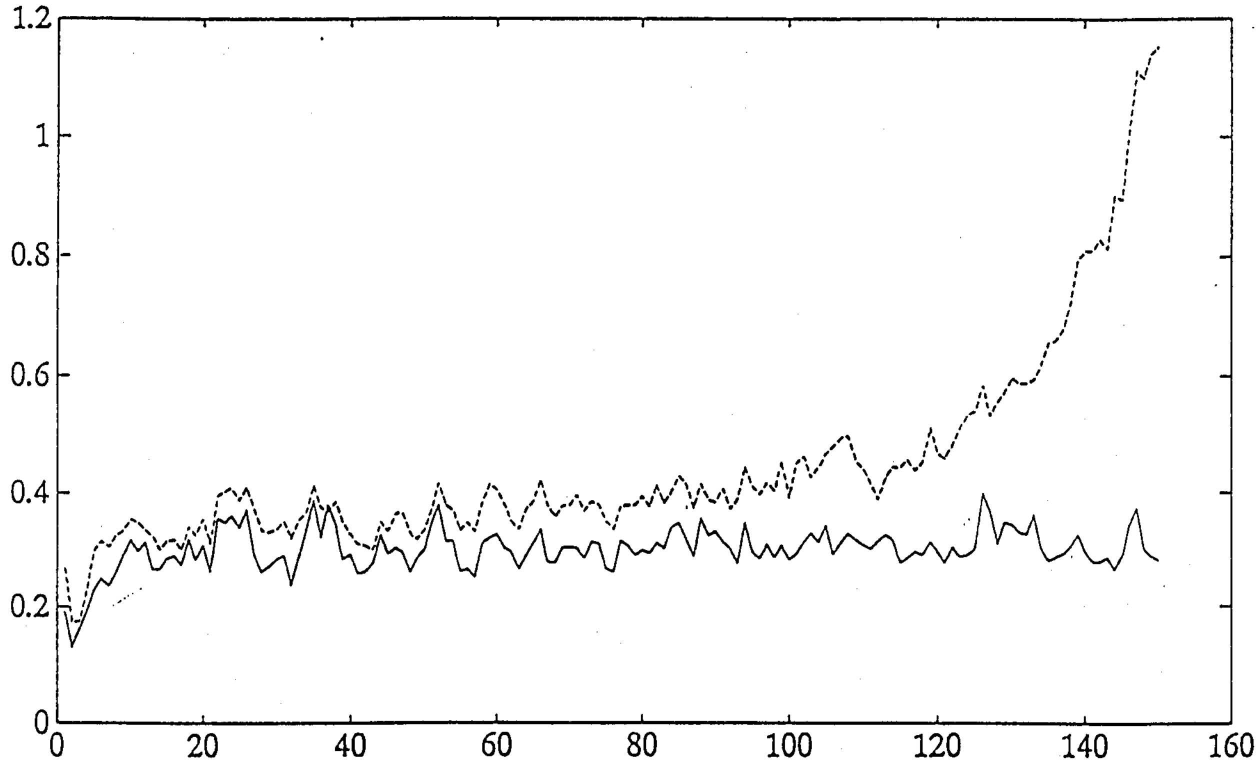


Figure 4.3: RMSEs versus time of NFWA (—) and NF(---) for Example 1.

extend the RNN output range. The EKF and IEKF fail to track the signal. The RMSEs averaged over 500 runs of the NFWA (—) and the NF (---) are plotted versus time in Fig. 4.3. The RNN in both cases is an MLPWIN with a single hidden layer of 9 fully interconnected neurons. The length of each training data sequence is 100. Both methods were tested for 150 time points to assess their ability to generalize. With the same MLPWIN architecture, the NFWA has a better performance than the NF, before 100 time points. After 100 time points, the latter deteriotes rapidly while the former remains good.

4.3. A Differencer as a Range Reducer

If the measurement process $y(t)$ consists of the vector-values, at discrete time points, of a continuous continuous-time process, then the vector value $y(t-1)$ is a reasonably good approximate of the vector value $y(t)$. This observation motivated a simple, yet effective way to reduce the range of of the measurements, when two consecutive measurements, $y(t-1)$ and $y(t)$, are not too far apart.

Consider the neural filter depicted in Fig. 4.4. A differencer is concatenated at the input terminals of an RNN. At each time point t, the differencer subtracts the preceding measurement vector $y(t-1)$ from the current measurement vector $y(t)$ and feeds the difference $y(t)-y(t-1)$ to the input terminals of the RNN.

Notice that if $y(0)$ is available as a constant vector, the sequence, $y(\tau)-y(\tau-1)$, $\tau=1,2,\cdots,t$, can easily be used to construct the sequence, $y(\tau)$, $\tau=1,2,\cdots,t$ and vice versa. Therefore, they contain the same amount of information about the signal $x(t)$, for $t=1,2,\cdots,T$. Since the constant vector is expected to be "memorized" in the RNN during its training, the differencer is not expected to destroy any information that the RNN can not salvage. This expectation is borne out by the following example, in which a differencer is concatenated at the input terminals of an MLPWIN and an EKF is used to reduce the output range of the MLPWIN. The neural filter consisting of the MLPWIN, the differencer and the EKF is depicted in FIG. 4.5.

We note here that a differencer concatenated at the input terminals of an RNN does not require additional treatment in a training algorithm for the rest of the neural filter except replacing $y(t)$ by $y(t)-y(t-1)$ as the input vector at time t.

Example 2. We tested the neural filter depicted in FIG. 4.5 using the signal and measurement processes described by the model:

$$\begin{aligned} x(t+1) &= x(t)+1.2\sin x(t)+1.21+0.2\xi(t), && (4.12)\\ y(t) &= x(t)+\varepsilon(t), && (4.13) \end{aligned}$$

where $\xi(t)$ and $\varepsilon(t)$ are independent standard white Gaussian sequences, and $x(0)=0$. Note that EKF should be a good estimator for this problem because the signal $x(t)$ enters the measurement $y(t)$ in a linear manner.

The RMSEs averaged over 500 runs are plotted versus time for the neural filter (—) and the EKF (---) in Fig. 4.6. The RNN in the neural filter is an MLPWIN with a single hidden layer of 9 fully interconnected neurons. The length of each training data sequence is 100. The neural filter is tested for 150 time points to assess its generalization capability. The performance of the neural filter is significantly better than that of the EKF.

4.4. Training of an RNN with a Range Extender

The concatenation of a range extender to an outward output node of an RNN necessitates special treatment in a training algorithm for the RNN.

We consider an MLPWIN with $L+1$ layers of nodes/neurons. The input layer is the 0th layer and the ouput layer is the Lth layer. The number of neurons in

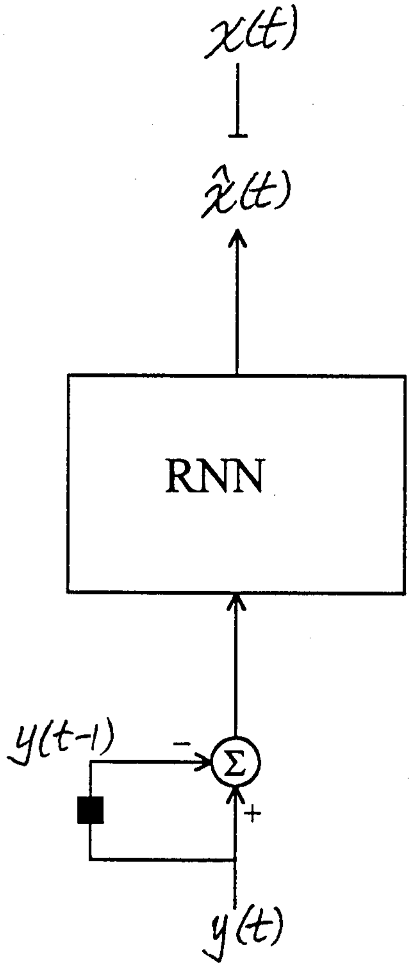


Figure 4.4: Neural filter with a differencer

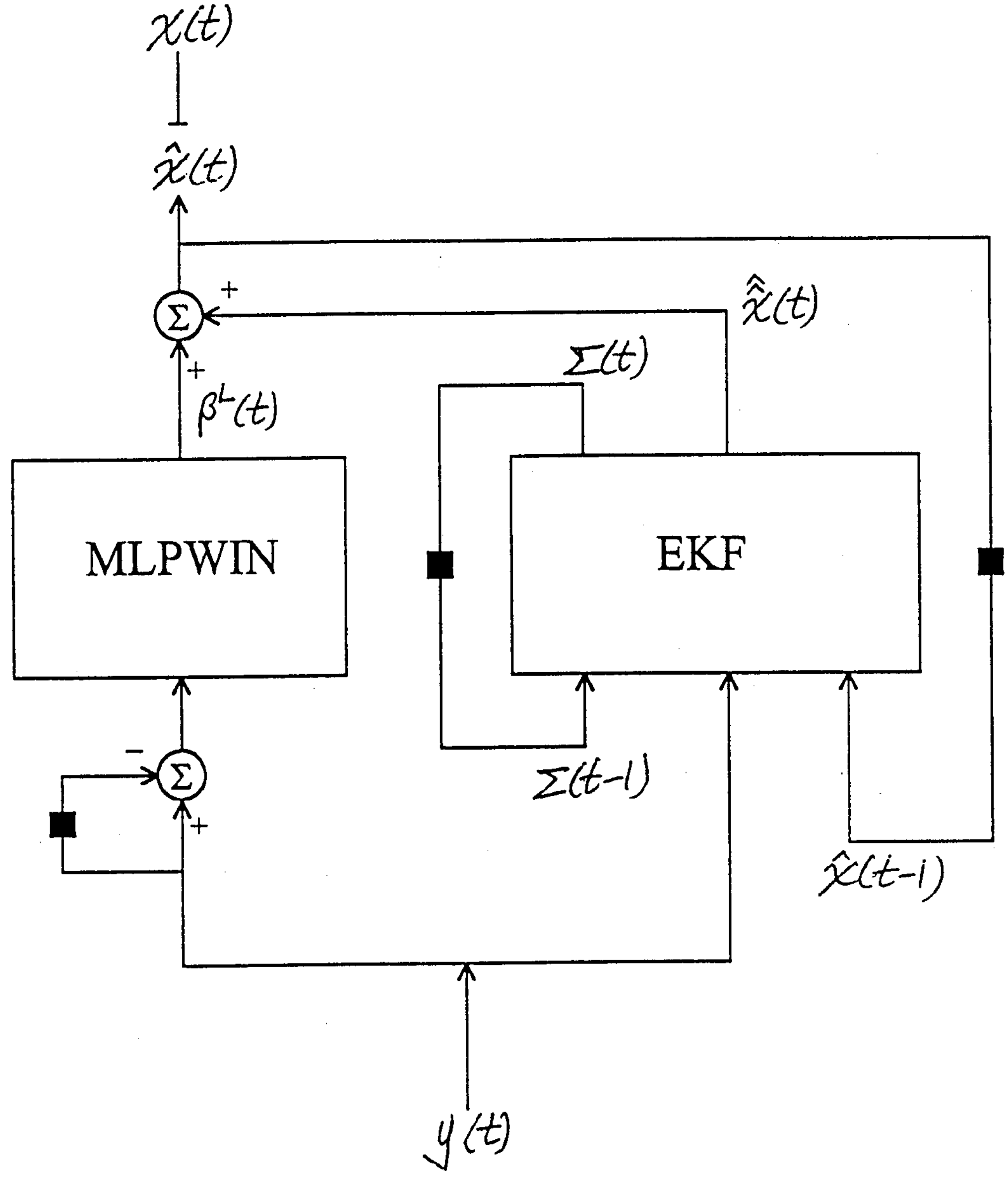


Figure 4.5: Neural filter with an EKF and a differencer

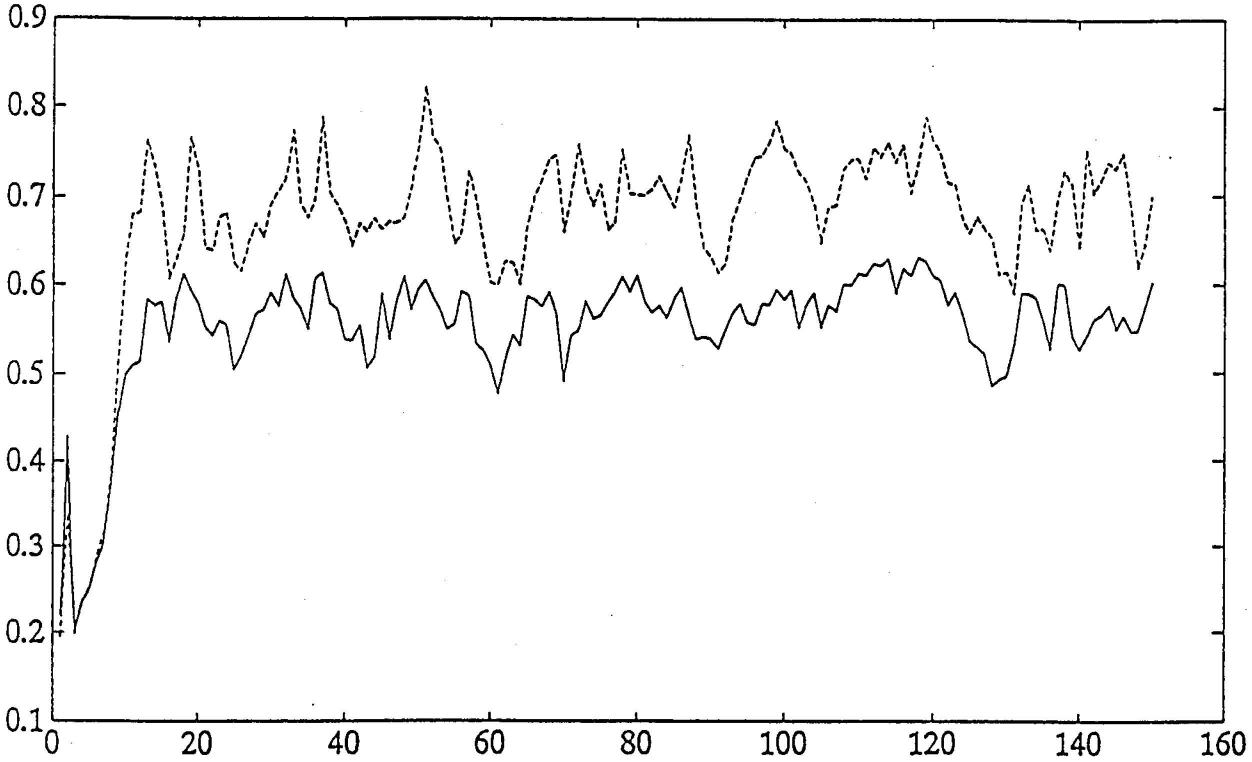


Figure 4.6: RMSEs of neural filter (—) and EKF (---) versus time for Example 2

the lth layer is n_l. The activation and weighted sum of the ith neuron in the lth layer at time t are denoted $\beta_i^l(t)$ and $\eta_i^l(t)$ respectively. The activation function of the ith neuron in the lth layer is denoted $a_i^l(\cdot)$. The weight for feeding $\beta_j^{l-1}(t)$ to the ith neuron in the lth layer at time t is denoted w_{ij}^l. The weight for feeding $\beta_j^l(t-1)$ to the ith neuron in the lth layer at time t is denoted w_{ij}^{rl}. Without loss of generality, we assume this MLPWIN has a single input node and a single output node.

The training data consists of pairing of measurement $y(t,\omega)$ and signal $x(t,\omega)$ for $t = 1, 2, \cdots, T$, where $\omega \in S$ and S is a collection of typical realizations of the measurement and signal process. For the realization ω, the activation and weighted sum of the ith neuron in the lth layer are denoted as $\beta_i^l(t,\omega)$ and $\eta_i^l(t,\omega)$ respectively. Obviously, $\beta_1^0(t,\omega) = y(t,\omega)$.

The cost function to be minimized in training is,

$$C = \frac{1}{T(\#S)} \sum_{\omega \in S} \sum_{t=1}^{T} (x(t,\omega) - \hat{x}(t,\omega))^2, \tag{4.14}$$

where $\hat{x}(t,\omega)$ is the estimate of $x(t,\omega)$ by the neural filter. C may be written as $C = (1/(\#S)) \sum_{\omega \in S} C_\omega$, where $C_\omega = (1/T) \sum_{t=1}^{T} (x(t,\omega) - \hat{x}(t,\omega))^2$.

To minimize C, dC/dw_{pq}^k and dC/dw_{pq}^{rk} are usually needed. We use BPTT algorithm, which gives,

$$\frac{dC}{dw_{pq}^k} = \frac{1}{(\#S)} \sum_{\omega \in S} \sum_{t=1}^{T} \frac{dC_\omega}{d\eta_p^k(t,\omega)} \beta_q^{k-1}(t,\omega), \tag{4.15}$$

and,

$$\frac{dC}{dw_{pq}^{rk}} = \frac{1}{(\#S)} \sum_{\omega \in S} \sum_{t=1}^{T} \frac{dC_\omega}{d\eta_p^k(t,\omega)} \beta_q^k(t-1,\omega). \tag{4.16}$$

$dC_\omega / d\eta_i^l(t,\omega)$ is obtained by recursive relation,

$$\frac{dC_\omega}{d\eta_i^l(t,\omega)} = \left(\sum_j w_{ji}^{l+1} \frac{dC_\omega}{d\eta_j^{l+1}(t,\omega)} + \sum_j w_{ji}^{rl} \frac{dC_\omega}{d\eta_j^l(t+1,\omega)}\right)(a_i^l)'(\eta_i^l(t,\omega)), \tag{4.17}$$

for $l = L-1, L-2, \cdots, 1$, $t = T, T-1, \cdots, 1$ and $\omega \in S$. The final condition is $dC/d\eta_i^l(T+1) = 0$.

4.4.1. Training of an RNN with an accumulator

With an accumulator as a range extender,

$$\hat{x}(t,\omega) = \beta_1^L(t,\omega) + \hat{x}(t-1,\omega), \tag{4.18}$$

for $t = 1, 2, \cdots, T$ and $\omega \in S$. $\hat{x}(0,\omega)$ is the a priori estimate of $x(0,\omega)$. The recursive formulas for obtaining $dC_\omega / d\eta_1^L(t,\omega)$ are,

$$\frac{dC_\omega}{d\eta_1^L(t,\omega)} = \frac{dC_\omega}{d\hat{x}(t,\omega)}(a_1^L)'(\eta_1^L(t,\omega)), \tag{4.19}$$

$$\frac{dC_\omega}{d\hat{x}(t,\omega)} = \frac{\partial C_\omega}{\partial \hat{x}(t,\omega)} + \frac{dC_\omega}{d\hat{x}(t+1,\omega)}, \tag{4.20}$$

for $t = T, T-1, \cdots, 1$ and $\omega \in S$. The final condition is $dC_\omega / d\hat{x}(T+1,\omega) = 0$.

4.4.2. Training of an RNN with an EKF

With an EKF and an adder as a range extender,

$$\hat{x}(t,\omega) = \beta_1^L(t,\omega) + \hat{x}(t,\omega), \tag{4.21}$$

for $t = 1, 2, \cdots, T$ and $\omega \in S$, where $\hat{x}(t,\omega)$ is the estimate generated by an EKF, which is a function of $y(t,\omega)$, $\hat{x}(t-1,\omega)$ and $\Sigma(t-1,\omega)$. Also note that $\Sigma(t,\omega)$ is a function of $\hat{x}(t-1,\omega)$ and $\Sigma(t-1,\omega)$. The recursive formulas for obtaining $dC_\omega / d\eta_1^L(t,\omega)$ are,

$$\frac{dC_\omega}{d\eta_1^L(t,\omega)} = \frac{dC_\omega}{d\hat{x}(t,\omega)}(a_1^L)'(\eta_1^L(t,\omega)), \tag{4.22}$$

$$\frac{dC_\omega}{d\hat{x}(t,\omega)} = \frac{\partial C_\omega}{\partial \hat{x}(t,\omega)} + \frac{dC_\omega}{d\hat{x}(t+1,\omega)} \frac{\partial \hat{\hat{x}}(t+1,\omega)}{\partial \hat{x}(t,\omega)} + \frac{dC_\omega}{d\Sigma(t+1,\omega)} \frac{\partial \Sigma(t+1,\omega)}{\partial \hat{x}(t,\omega)}, \tag{4.23}$$

$$\frac{dC_\omega}{d\Sigma(t,\omega)} = \frac{dC_\omega}{d\hat{x}(t+1,\omega)} \frac{\partial \hat{\hat{x}}(t+1,\omega)}{\partial \Sigma(t,\omega)} + \frac{dC_\omega}{d\Sigma(t+1,\omega)} \frac{\partial \Sigma(t+1,\omega)}{\partial \Sigma(t,\omega)}, \tag{4.24}$$

for $t = T, T-1, \cdots, 1$ and $\omega \in S$. The final conditions are $dC_\omega / d\hat{x}(T+1,\omega) = dC_\omega / d\Sigma(T+1,\omega) = 0$.

5. Adaptive Neural Filtering

In the conventional theory of adaptive filtering, the system model contains a parameter process $\theta(t)$, which represents the changes in the filtering environment. The parameter process $\theta(t)$ may either be an unknown deterministic function of time or a stochastic process with known defining joint probabilities. Given the value of $\theta(t)$ at each t, the system model is a well-defined stochatic system (i.e. all the defining joint probabilities for the signal and measurements are determined).

In the context of neural filtering, a system model may or may not be available and the underlying system is considered instead. However, the changes in the filtering environment can still be represented by a parameter process $\theta(t)$. As before, $\theta(t)$ may be an unknown deterministic function of time or a stochastic process with known defining joint probabilities. But $\theta(t)$ may also be a stochastic process with unknown defining joint probabilities.

The difference between the definitions of an unknown deterministic function and a stochastic process with unknown defining joint probabilities is vague, because hypothetically, any process can be observed over a long enough time for a sufficient number of times in typical working environments so that the observed realizations define the process as a stochastic process (with all its defining joint probabilities). Hence, an unknown process should be viewed as an unknown deterministic function, unless the variability in its hypothetical probability distribution is small enough for this distribution to be useful.

Whether $\theta(t)$ is treated as an unknown deterministic process or a stochastic process with known or unknown statistics, two first questions to ask before considering adaptive neural filtering are the following:

(a). Can we simply use a recurrent neural network (RNN), that inputs the measurement process and outputs the estimate of the signal process, and let the RNN learn in its training how to deal with $\theta(t)$ internally?

(b). Can we train and use an RNN to estimate $\theta(t)$?

We notice that in training either of the two RNNs mentioned in these questions, some training data set, containing different actual or simulated, hidden or explicit realizations of $\theta(t)$, has to be used. The trained RNN is supposed to be optimal for the distribution of $\theta(t)$ represented by these realizations. If $\theta(t)$ is a stochastic process with reasonably small variability and the training data set sufficiently represents this stochastic process, the answers to both of these questions are possitive and no adaptive filtering is necessary. However, if $\theta(t)$ is either an unknown deterministic process or a stochastic process with large variability, which is not fully represented by the training data set, the trained RNN may be far from being optimal in a certain usage of the trained RNN. Moreover, the size of the RNN may have to be economically unacceptably large to accomodate the large variability of the distribution of a stochastic $\theta(t)$. Therefore, if $\theta(t)$ is an unknown deterministic process or a stochastic process with large variability, the answer to the above two questions are negative. In this case, the filtering for the signal $x(t)$ has to be adapted to the changing $\theta(t)$ on the fly, meaning that the estimation for $\theta(t)$ at a current time t has to be made completely or essentially on the basis of the recurrent measurements $y(s), s \leq t$. Basically, no a priori knowledge about $\theta(t)$ in either the form of statistics or the form of signal and measurement realizations can be used. This situation actually defines the word "adaptation" in the context of neural filtering.

For the simplicity of our discussion, we will restrict it to parameter functions that are unknown deterministic in the sequel. Only constant parameter functions will be considered. Two methods of adaptive filtering will be presented. These methods evaluate the maximum likelihood estimate of $\theta(t)$ and the minimum variance estimate of $x(t)$, given the measurements y^t.

In the rest of this Chapter, we assume that the signal $x(t)$ and measurement $y(t)$ satisfy the discrete-time equation,

$$\begin{aligned} x(t+1,\theta^{t+1}) &= f(x(t,\theta^t),\theta(t+1)) + G(x(t,\theta^t),\theta(t+1))w(t), && (5.1)\\ y(t,\theta^t) &= h(x(t,\theta^t),\theta(t)) + v(t), && (5.2) \end{aligned}$$

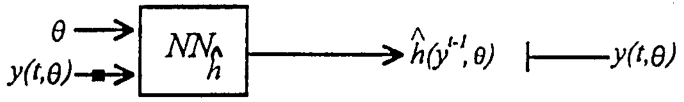
where f, G, and h may or may not be known; and $w(t)$ and $v(t)$ are independent white Gaussian sequences with mean zero and variances $E[w(t)w'(t)] = Q(t)$ and $E[v(t)v'(t)] = R(t)$.

5.1. Adapting to Unknown Constant Parameters

For a given value of θ^t, estimating $x(t)$ given y^t is a standard filtering problem. But, θ^t is not given and $\theta(t)$ has to be estimated on the fly. However, a neural filter can be synthesized in advance (i.e. before the filter is put to use) that processes the estimate $\hat{\theta}(t)$ of $\theta(t)$ as well as $y(t)$ as measurements, and produces a minimum variance estimate $\hat{x}(t)$ of $x(t)$ for these measurements. The question is how best to estimate $\theta(t)$ given y^t on the fly. under the assumption $\theta(t)$ is a constant θ, we will illustrate each of 2 adaptive filtering methods by first describing how each component is synthesized, then showing how they are interconnected, and finally how θ is estimated on the fly by minimizing an error criterion. The interpretations of these two methods will then be given.

Method 1

(a). Train $NN_{\hat{h}}$ as follows:



(5.3)

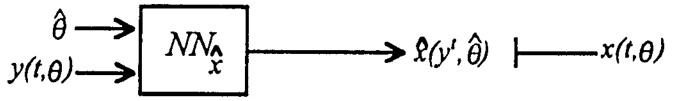
where θ ranges over all possible values of θ.

(b). Pick a possible value of θ and simulate (5.1), (5.2) and $NN_{\hat{h}}$ to get $y(t,\theta)$ and $\hat{h}(y^{t-1},\theta)$, for $t = S, S+1, \cdots$, where S is a time at which $y(t,\theta)$ is in a steady state. Get an estimate $\hat{\theta}$ of θ by minimizing

$$C(\theta) = \frac{1}{T} \sum_{t=S}^{S+T-1} [y(t,\theta) - \hat{h}(y^{t-1},\theta)]' R^{-1}(t) [y(t,\theta) - \hat{h}(y^{t-1},\theta)]$$

where T is the length of time which is sufficient for a good estimation of θ. Repeat above to obtain a sufficient number of $\{(y(t,\theta), x(t,\theta), \hat{\theta}), t = S, S+1, \cdots\}$ as training data for the next step.

(c). Train $NN_{\hat{x}}$ as follows:



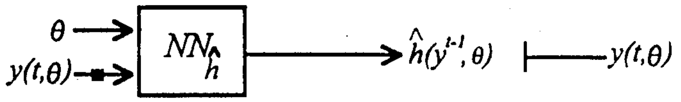
(5.4)

(d). After training, (5.3) and (5.4) are run in parallel with $\hat{\theta}$ determined at t by minimizing

$$C(\theta) = \frac{1}{T} \sum_{\tau=t-T+1}^{t} [y(\tau,\theta) - \hat{h}(y^{\tau-1},\theta)]' R^{-1}(\tau) [y(\tau,\theta) - \hat{h}(y^{\tau-1},\theta)].$$

Method 2

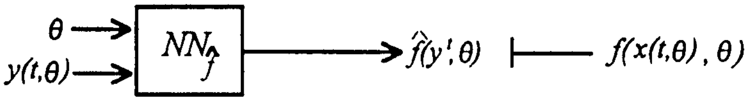
(a). Train $NN_{\hat{h}}$ as follows:



(5.5)

where θ ranges over all possible values of θ.

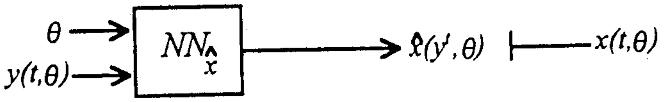
(b). Train $NN_{\hat{f}}$ as follows:



(5.6)

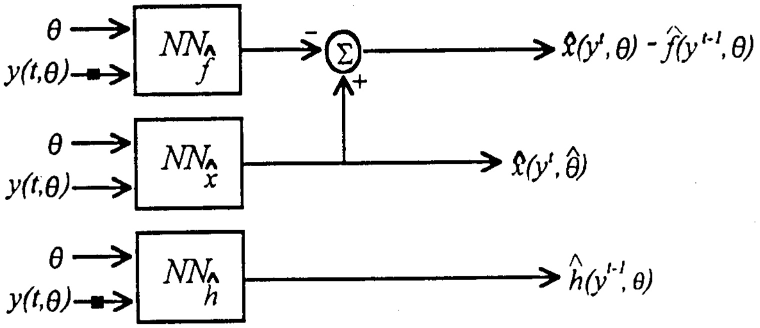
where $f(x(t,\theta),\theta)$ can be replaced by $x(t+1,\theta)$, if f is unknown.

(c). Train $NN_{\hat{x}}$ as follows:



(5.7)

(d). The above 3 RNNs being connected as follows:


θ
y(t,θ)
$NN_{\hat{f}}$
θ
y(t,θ)
$NN_{\hat{x}}$
θ
y(t,θ)
$NN_{\hat{h}}$
Σ
-
+
$\hat{x}(y^t,\theta) - \hat{f}(y^{t-1},\theta)$
$\hat{x}(y^t,\hat{\theta})$
$\hat{h}(y^{t-1},\theta)$

(5.8)

the estimation error covariance,

$$V(\theta) := E[\varepsilon(\theta)\varepsilon'(\theta)],$$
$$\varepsilon(\theta) := [\hat{x}'(y^t,\theta) - \hat{f}'(y^{t-1},\theta), y'(t,\theta) - \hat{h}'(y^{t-1},\theta)]',$$

is approximated for each possible value of θ by the Monte Carlo method using $NN_{\hat{h}}$ and $NN_{\hat{f}}$ jointly. An MLP may serve as a good approximating function here.

(e). An estimate $\hat{\theta}$ of θ is obtained by minimizing

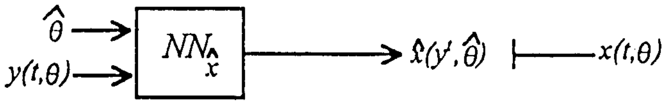
$$D(\theta) := \frac{1}{T}\sum_{\tau=t-T+1}^{t}[\varepsilon(t,\theta)V^{-1}(\theta)\varepsilon'(t,\theta)], \tag{5.9}$$

where

$$\varepsilon(t,\theta) := [\hat{x}'(y^t,\theta) - \hat{f}'(y^{t-1},\theta), y(t,\theta) - \hat{h}(y^{t-1},\theta)]'.$$

(f). To retrain $NN_{\hat{x}}$, the retraining data is collected as follows: For a possible value of θ, simulate (5.1) and (5.2), run (5.8) and minimize (5.9) to get $\hat{\theta}$, then $((\hat{\theta}, y(t,\theta)), x(t,\theta))$ is used as a training input/output pair in retraining

$NN_{\hat{x}}$ as follows:



(5.10)

(g). Repeat (e) and (f) using the latest version of $NN_{\hat{x}}$, until $NN_{\hat{x}}$ converges.

Interpretation of Method 1

Recall the known property of the innovation process, $\mathcal{I}(t,\theta) := y(t,\theta) - E[y(t,\theta)|y^{t-1}]$, that for each θ, $\mathcal{I}(t,\theta)$ is a white Gaussian process with mean 0 and covariance R. Minimizing $C(\theta)$ is equivalent to maximizing the conditional likelihood function for θ given the measurements y^{t-1}. Therefore, Method 1 produce a maximum likelihood estimate $\hat{\theta}$ of θ and the corresponding minimum variance estimate $\hat{x}(y^t,\hat{\theta})$ for y^t and $\hat{\theta}$.

Interpretation of Method 2

The key question here is "What is the conditional probability distribution of $E[x(t,\theta)|y^t] - E[x(t,\theta)|y^{t-1}]$ give y^t?" If the system, (5.1) and (5.2), is linear, i.e. $f(x(t,\theta),\theta) = F(t,\theta)x(t,\theta)$, $G(x(t,\theta),\theta) = G(t,\theta)$, and $h(x(t,\theta),\theta) = H(t,\theta)x(t,\theta)$, the measurement update equation is

$$\begin{aligned} E[x(t,\theta)|y^t] = E[x(t,\theta)|y^{t-1}] + \Sigma_{t|t-1}H_t'(H_t'\Sigma_{t|t-1}H_t + R)^{-1}[y(t,\theta) \\ -H_t'E[x(t,\theta)|y^{t-1}], \\ \Sigma_{t|t} = \Sigma_{t|t-1} - \Sigma_{t|t-1}H_t(H_t'\Sigma_{t|t-1}H_t + R)^{-1}H_t'\Sigma_{t|t-1}, \\ H_t = H(t,\theta), \end{aligned}$$

where

$$\begin{aligned} E[x(t,\theta)|y^t] - E[x(t,\theta)|y^{t-1}] = K(t)\mathcal{I}(t,\theta) \\ K(t) = \Sigma_{t|t-1}H_t'(H_t'\Sigma_{t|t-1}H_t + R)^{-1} \end{aligned}$$

is a white Gaussian process with mean 0 and covariance $K(t)R(t)K'(t)$.

Although in general, $E[x(t,\theta)|y^t] - E[x(t,\theta)|y^{t-1}]$ is not Gaussian, the measurement update equation of the extended Kalman filter for the system, (5.1) and (5.2), does provide a reasonable approximation of the quantity. Hence it is approximately white Gaussian with mean 0 and covariance $K(t)R(t)K'(t)$, where

$$H_t' = \left. \frac{\partial h(x,\theta)}{\partial x} \right|_{x=E[x(t,\theta)|y^{t-1}]} .$$

In view of this approximation, Method 2 provides approximately a maximum likelihood estimate of θ and a minimum variance estimate of x.

5.2. Adapting to Unknown Dynamic Parameters

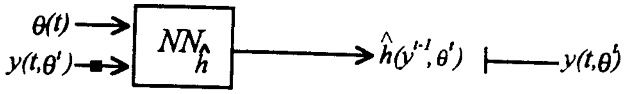
In addition to (5.1) and (5.2), we assume, in this section, that the parameter function $\theta(t)$ is a dynamic process satisfying the dynamic equation,

$$\theta(t+1) = p(\theta(t), \theta(t-1), ..., \theta(t-m+1), u(t)), \qquad (5.11)$$

where m is an unknown integer, p is an unknown function and $u(t)$ is an unknown stochastic process independent of w and v. The variabilities of p, m, and u from application to application are assumed to be so large that the information carried by the statistics of the stochastic process $\theta(t)$ is not sufficient for synthesizing a neural filter in advance to estimate $\theta(t)$ or to estimate $x(t)$ without adapting to $\theta(t)$. We note in passing that when the dynamic equation (5.11) takes on the special form, $\theta(t+1) = \theta(t)$, the adaptive filtering problem considered here is the same as that considered in the preceding section.

Method 3

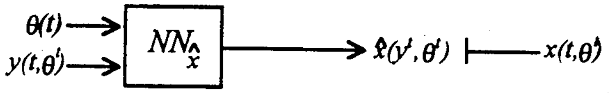
(a). Train $NN_{\hat{h}}$ as follows:



(5.12)

where θ ranges over all possible values of θ.

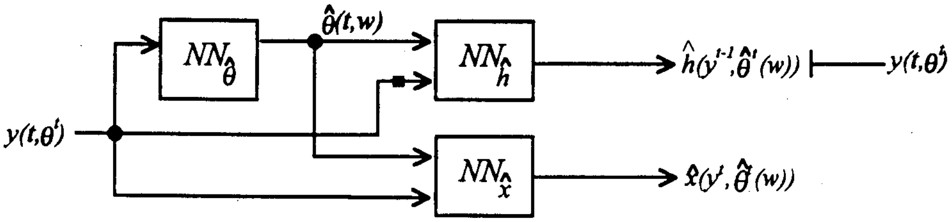
(b). Train $NN_{\hat{x}}$ as follows:



(5.13)

(c). After training, (5.12) and (5.13) are run in parallel as follows with another RNN, $NN_{\hat{\theta}}$, whose weights, w, are adjusted by minimizing

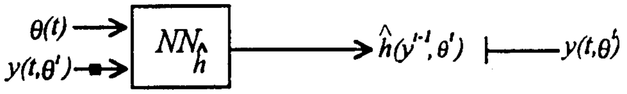
$$C(w) = \frac{1}{T} \sum_{\tau=t-T+1}^{t} [y(\tau,\theta^\tau) - \hat{h}(y^{\tau-1},\hat{\theta}^\tau(w))]' R^{-1} [y(\tau,\theta^\tau) - \hat{h}(y^{\tau-1},\hat{\theta}^\tau(w))],$$



(5.14)

Method 4

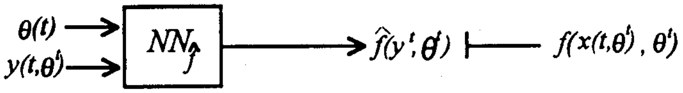
(a). Train $NN_{\hat{h}}$ as follows:



(5.15)

where a large variety of the parameter functions $\theta(t)$ is used. The distribution of them should be such that given any parameter function, the trained $NN_{\hat{h}}$ produces the optimal estimate of $y(t, \theta^t)$ at any t.

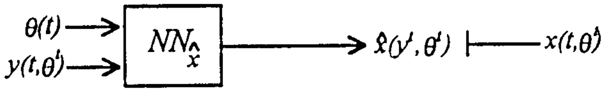
(b). Train $NN_{\hat{f}}$ as follows:



(5.16)

where $f(x(t, \theta), \theta)$ can be replaced by $x(t + 1, \theta)$, if f is unknown.

(c). Train $NN_{\hat{x}}$ as follows:



(5.17)

(d). The above 3 RNNs are run as follows with another RNN, $NN_{\hat{\theta}}$, whose weights, w, are adjusted by minimizing

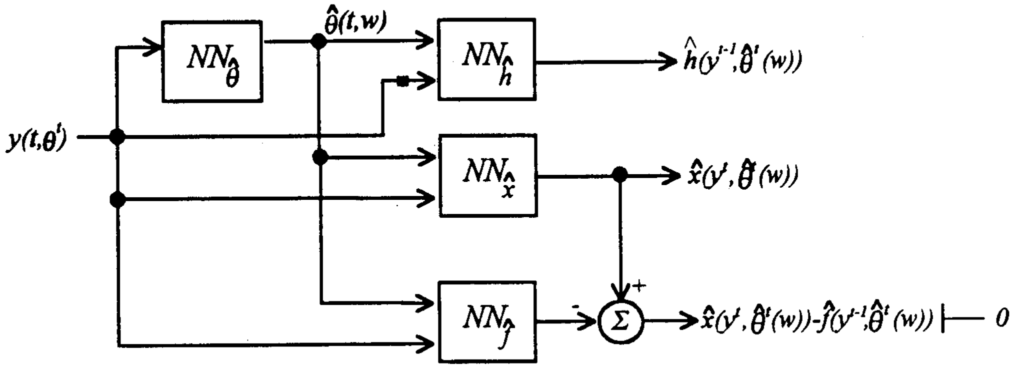
$$D(w) := \frac{1}{T} \sum_{\tau=t-T+1}^{t} [\varepsilon(\tau, w) V^{-1}(\tau) \varepsilon'(\tau, w)],$$

where

$$\varepsilon(t, w) := [\hat{x}'(y^t, \hat{\theta}^t(w)) - \hat{f}'(y^{t-1}, \hat{\theta}(w)), y(t, \theta) - \hat{h}(y^{t-1}, \hat{\theta}^t(w))]'$$

and

$$V(t) = \begin{bmatrix} K(t)R(t)K'(t) & 0 \\ 0 & R(t) \end{bmatrix}$$


$\hat{\theta}(t,w)$
$NN_{\hat{\theta}}$
$NN_{\hat{h}}$
$\hat{h}(y^{t-1},\hat{\theta}^t(w))$
$y(t,\theta^t)$
$NN_{\hat{x}}$
$\hat{x}(y^t,\hat{\theta}^t(w))$
$NN_{\hat{f}}$
-
+
Σ
$\hat{x}(y^t,\hat{\theta}^t(w))-\hat{f}(y^{t-1},\hat{\theta}^t(w))$
0

(5.18)

5.3. Numerical Results

The signal and measurement equations used in our numerical study are

$$\begin{aligned} x(t+1) &= \max\{\theta x(t)(1-x(t)) + 0.1w(t), 0.0099\theta\}, \\ y(t) &= 4(x(t)-0.2)^3 + 0.02v(t), \end{aligned}$$

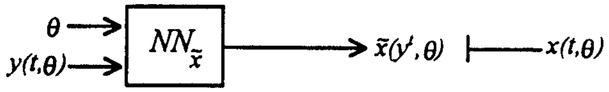
where $w(t)$ and $v(t)$ are independent standard white Gaussian sequences and θ is an unknown constant in the interval $[2, 3.6]$.

Only Method 1 was used in our numerical study. All the RNNs to be discussed have the architecture 2:7r:5:1. Let us recall the steps to synthesize an adaptive neural filter by Method 1. First, $NN_{\hat{h}}$ is synthesized. Secondly, $NN_{\hat{h}}$ is used to generate the training data set $\{(y(t, \theta^t), x(t, \theta^t), \hat{\theta}), t = s, s+1, ...\}$ for a given sliding window length T, as described in Step (b) of Method 1. Finally, this training data set is used to synthesize $NN_{\hat{x}}$.

The first adaptive neural filter that was synthesized for the foregoing system had a sliding window of length $T_1 = 1,000$. The histogram of $\hat{\theta} - \theta$ from $1,000$ trials of the adaptive neural filter is shown in Figure 5.1. The second adaptive neural filter synthesized in our numerical study had a sliding window of length $T_2 = 100$.

The histogram of $\hat{\theta} - \theta$ from $10,000$ trials is shown in Figure 5.2. Notice that both histograms are more or less Gaussian and unbiased, and as expected, the standard deviation in Figure 5.1 is about $\sqrt{T_2/T_1} = 1/\sqrt{10}$ of that in Figure 5.2.

To provide a benchmark or lower bound for the mean square error of the adaptive neural filter, an RNN, $NN_{\tilde{x}}$, is trained as follows



Notice that the true value θ is used as an input to $NN_{\tilde{x}}$. Thus, if $NN_{\tilde{x}}$ is properly synthesized, $\tilde{x}(y^t, \theta^t)$ is a better estimate of $x(t, \theta^t)$ than $\hat{x}(y^t, \hat{\theta}^t)$, which is produced by the adaptive neural filter, and the mean square error of $\tilde{x}(y^t, \theta^t)$ is a lower bound of that of $\hat{x}(y^t, \hat{\theta}^t)$.

The third adaptive neural filter synthesized in our numerical study had a sliding window of 20. The mean square error for 1.000 trials of the adaptive neural filter is shown in Figure 5.3 over a period of 20 time points. The difference between the mean square errors of $\tilde{x}(y^t, \theta^t)$ and $\hat{x}(y^t, \hat{\theta}^t)$ for 1,000 trials is shown in Figure 5.4. Notice that this difference is very small.

When the sliding window length was reduced to 10, this difference remains rather small. The mean square error for 1,000 trials of the adaptive neural filter and this difference between the mean square errors of $\tilde{x}(y^t, \theta^t)$ and $\hat{x}(y^t, \hat{\theta}^t)$ are shown in Figures 5.5 and 5.6, respectively.

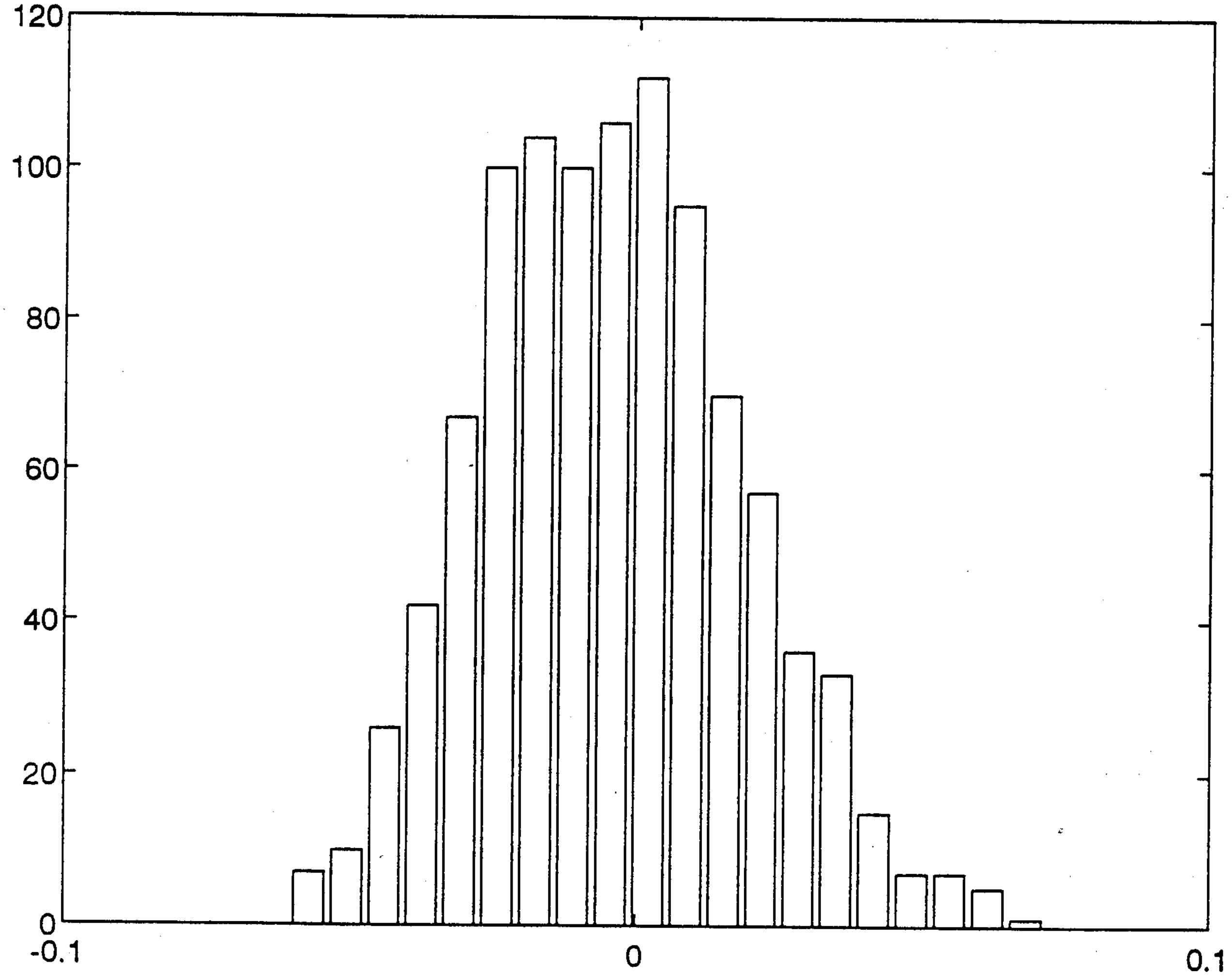


Figure 5.1: The histogram of $\hat{\theta} - \theta$ of the first adaptive neural filter

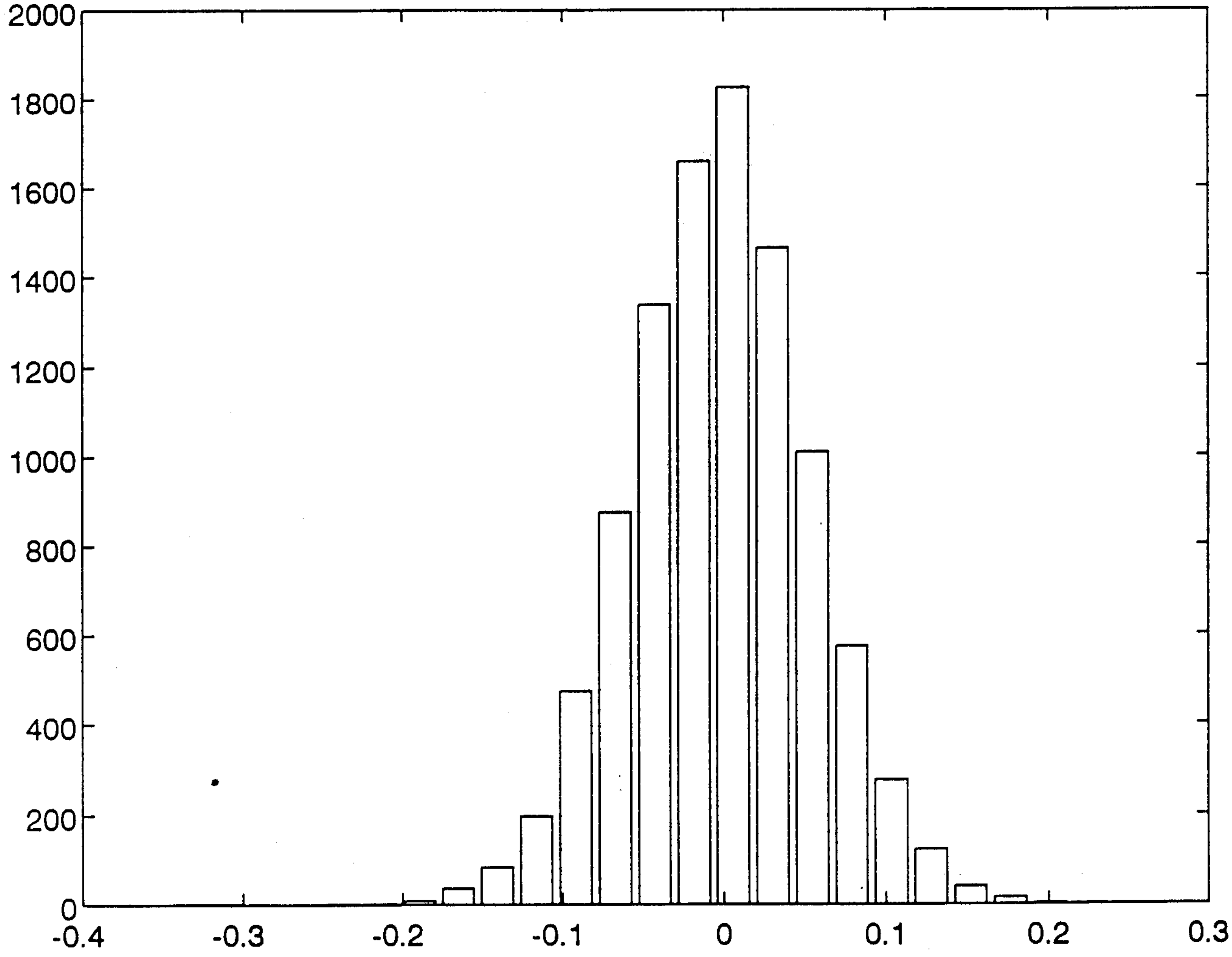


Figure 5.2: The histogram of $\hat{\theta} - \theta$ of the second adaptive neural filter

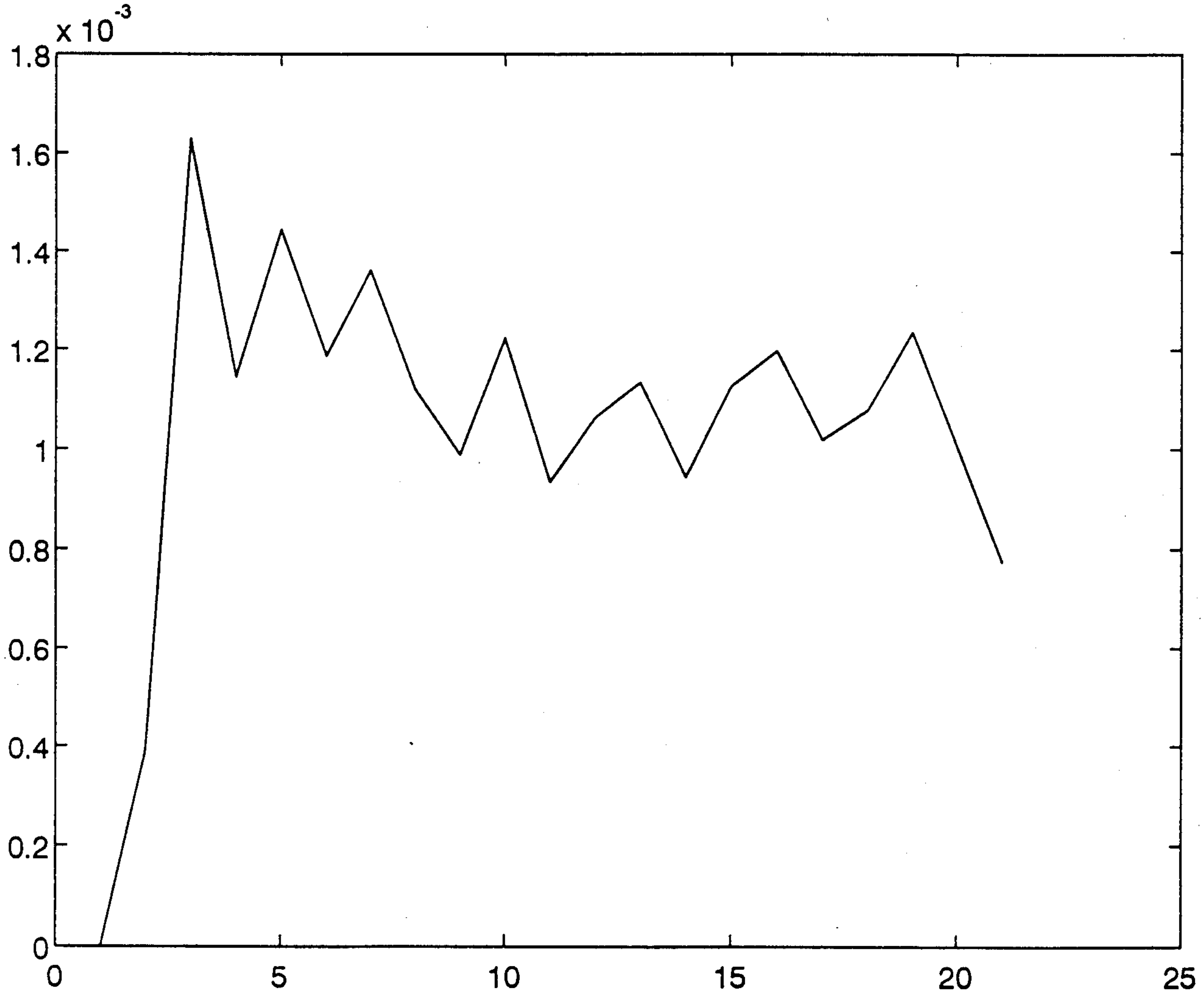


Figure 5.3: The mean square error of the third adaptive neural filter

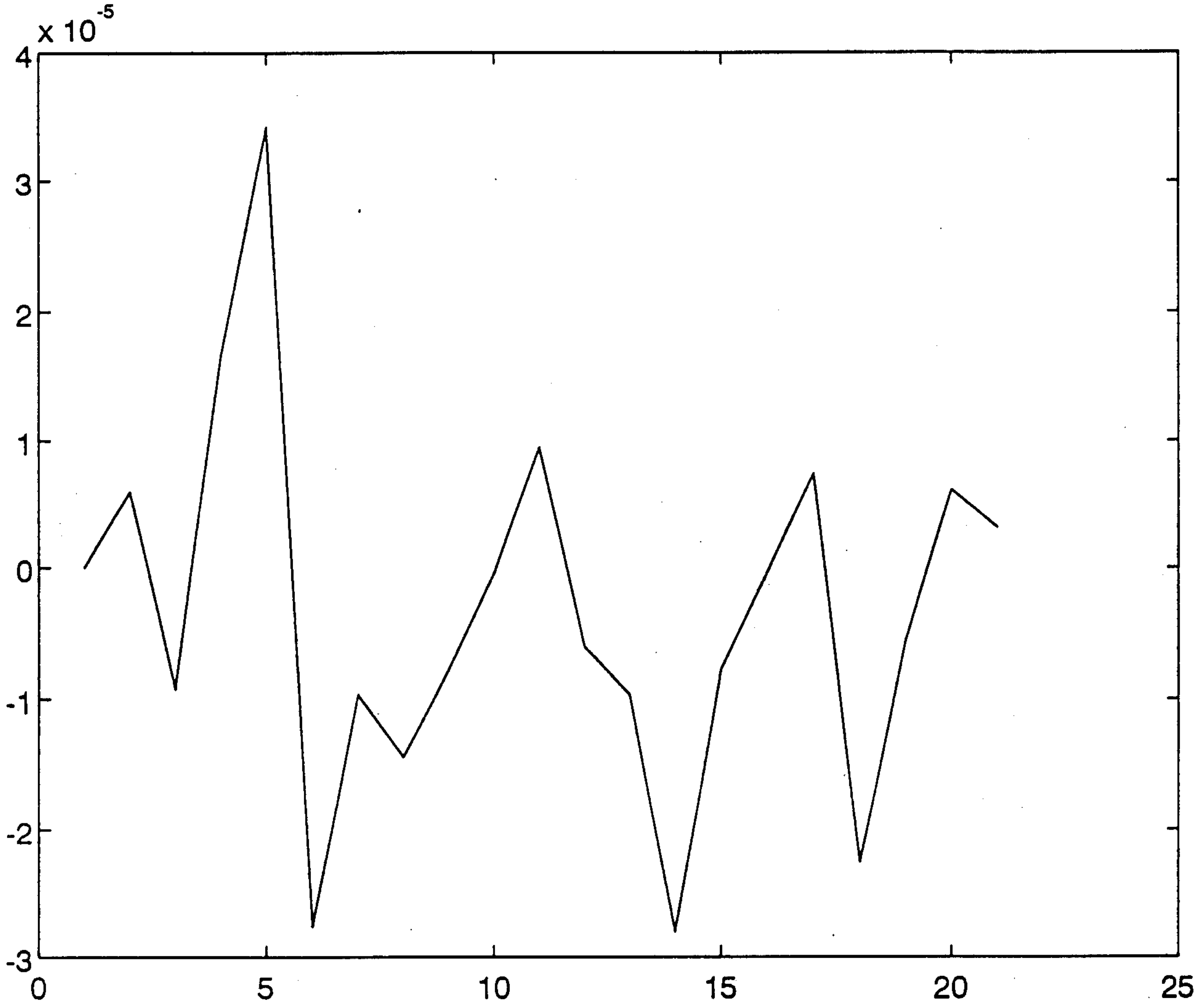


Figure 5.4: The difference between the mean square errors of $\tilde{x}(y^t, \theta^t)$ and $\hat{x}(y^t, \hat{\theta}^t)$ of the third adaptive neural filter

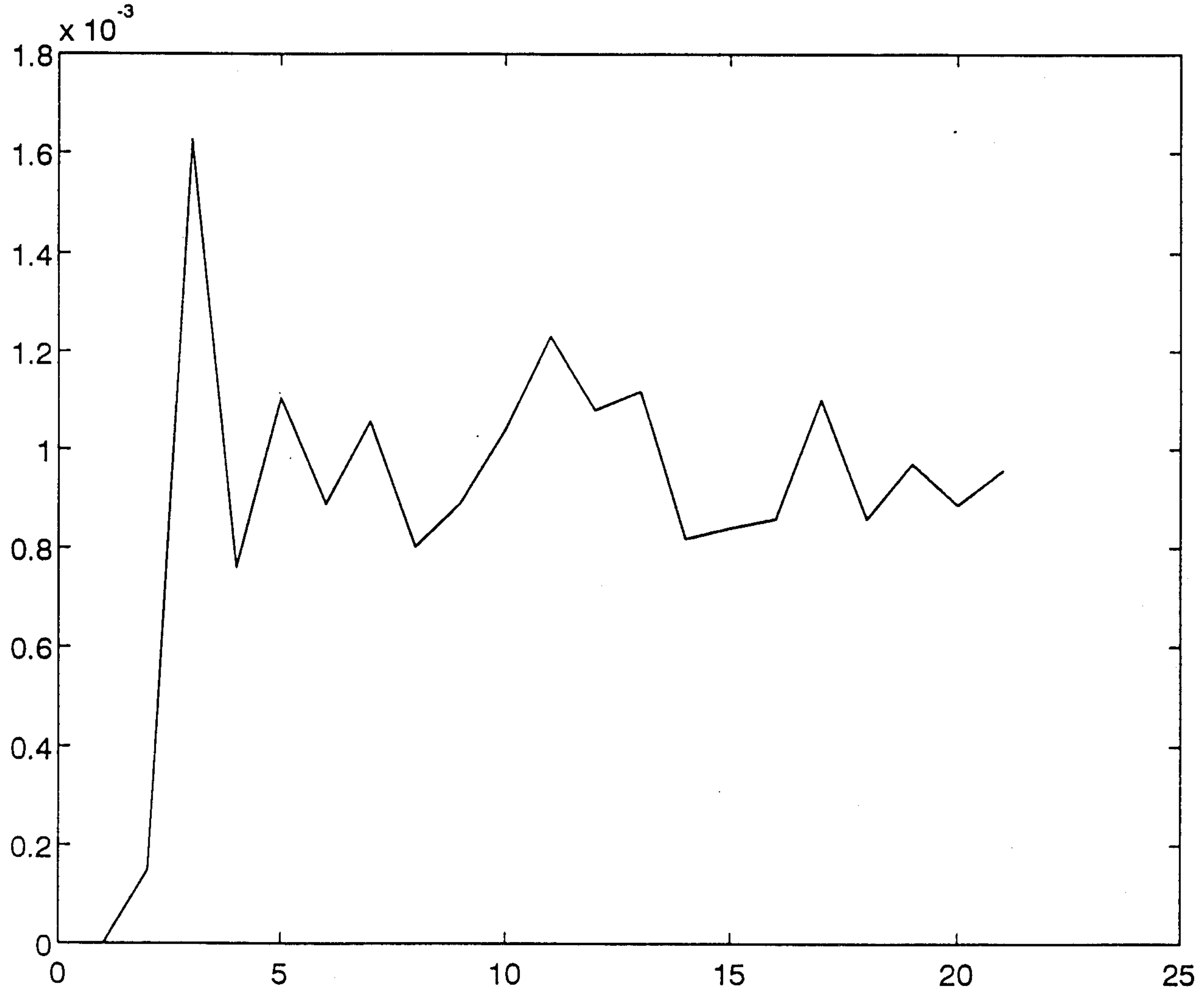


Figure 5.5: The mean square error of the adaptive filter when the sliding window length was reduced to 10

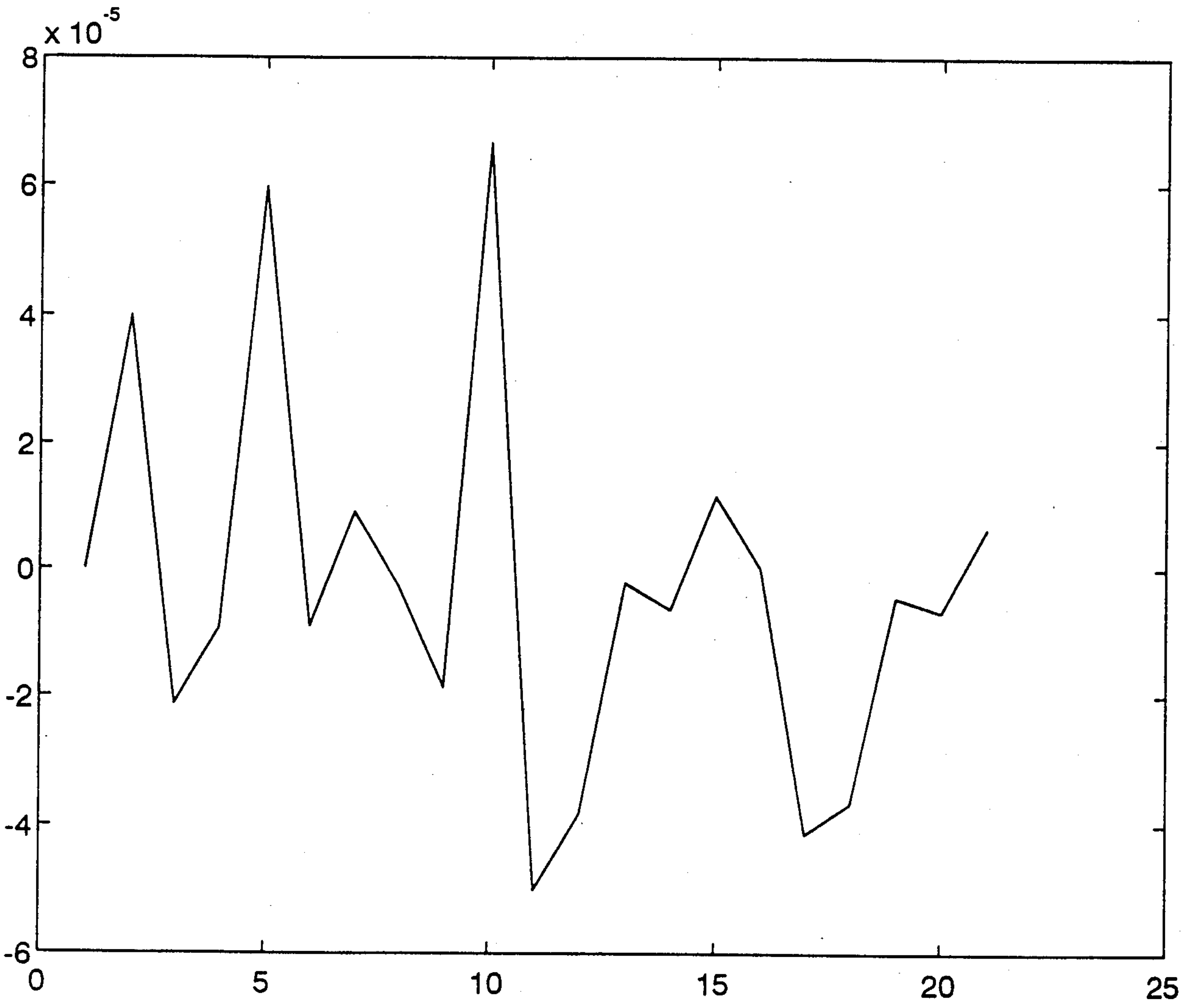


Figure 5.6: The difference between the mean square errors of $\tilde{x}(y^t, \theta^t)$ and $\hat{x}(y^t, \hat{\theta}^t)$ when the sliding window length was reduced to 10

6. Differentiating and Pruning Multilayer Feedforward Neural Networks

Two questions that arise after (or before) a multilayer feedforward network is trained up on a large amount of data are the following:

(a). How is a certain output (or response) variable related to a certain input (or independent) variable?

(b). Are some input variables superfluous? Which ones are they? Can we eliminate them?

To answer these questions, we need to find the structures of the network which serve to reveal heuristically meaningful functional characteristics hidden in layers of neurons and to appreciate the separate and joint effects produced on the response variables by changes in the independent variables.

Since the network training is actually a nonlinear regression[4], let us examine how the greatly successful linear regression model is used to interpret data. In the linear model, the parameters that summarize data are the rates of change of the response variables with respect to (w.r.t.) the independent variables. These first order derivatives of the response variables w.r.t. the input variables are precisely the heuristically meaningful structures that are used to explain data and select or deselect independent variables in the regression analysis. Similar structures do exist in a multilayer feedforward network.

A multilayer feedforward network is a multistage mixture of summations and compositions of logistic functions or hyperbolic tangents, which are analytic, i.e. infinitely differentiable and expandable into power series converging to the expanded functions. It is therefore also analytic. It is known that the derivatives of an analytic function w.r.t. its independent variables determine how much linear, quadratic, cubic and higher order components there are in the function.

It is proven[10] that a multilayer feedforward network with a single hidden layer and reasonably smooth activation function is capable of approximate an arbitrary function and its derivatives to any degree of accuracy. So it is justified to infer the dependence of the output variables on the input variables for the training data by differentiating the trained-up approximation network. Obviously, the magnitudes of these derivatives, especially those of the first order, do provide us with ample intuitive understanding of the function and guide us in deselecting independent variables.

In this paper, we present two methods of evaluating the derivatives of the output variables of a multilayer feedforward network w.r.t. the independent variables, not only of the first order but also of higher orders. In analogy to the backpropagation training [22, 26], one of our methods propagates the derivatives of the response variables w.r.t. neuron outputs backward one layer at a time. These derivatives (w.r.t. neuron outputs) can be used to interpret the relationships between neurons of different layers and to prune superfluous neurons as well as inputs. On the contrary, our second method propagates the derivatives of neuron outputs w.r.t. the input variables forward one layer at a time.

The two methods are equivalent when they are used to determine derivatives of the output variables w.r.t. the input variables. It is shown that they both result from a fundamental chain rule for differentiating a multilayer feedforwrd neural network, which is derived in the appendix. This chain rule is also used to differentiate the aforementioned derivatives w.r.t. the network weights.

In Sections 6.4–6.6 of the paper, we discuss how these derivatives are applied to pruning the superfluous weights and input variables: First the covariances of the derivatives and those of the weights are estimated; Then hypotheses are statistically tested to determine which inputs and/or weights are prunable. A numerical example is given in Section 6.7.

6.1. Differentiation by Forward Propagation

Consider a multilayer feedforward network with L layers. Let the weighted sum in and the output of the i-th neuron in the l-th layer be denoted by y_i^l and x_i^l, respectively. Thus x_i^0 and x_i^L are the i-th input and the i-th output. The outputs of the network are obtained by the forward propagation through each layer:

$$\begin{aligned} y_i^l &= \sum_j w_{ij}^l x_j^{l-1}, \\ x_i^l &= f(y_i^l), \end{aligned} \tag{6.1}$$

for $l = 1, \cdots, L$, where f is the neuron activation function and w_{ij}^l is the weight from the j-th neuron in the $(l-1)$-th layer to the i-th neuron in the l-th layer.

By the chain rule of differentiation,

$$\begin{aligned} \frac{\partial x_i^l}{\partial x_j^0} &= \frac{dx_i^l}{dy_i^l} \cdot \frac{\partial y_i^l}{\partial x_j^0} \\ &= f'(y_i^l) \cdot \frac{\partial y_i^l}{\partial x_j^0}, \end{aligned} \tag{6.2}$$

where,

$$\frac{\partial y_i^l}{\partial x_i^0} = \sum_k w_{ik}^l \cdot \frac{\partial x_k^{l-1}}{\partial x_j^0}.$$

Thus, given $\partial x_k^1/\partial x_j^0 = f'(y_k^1) \cdot w_{kj}^1$, we can recursively calculate $\partial x_i^l/\partial x_j^0$ for $l = 2, \cdots, L$.

For the second order derivatives, we need to utilize the propagated first order derivatives as follows:

$$
\begin{aligned}
\frac{\partial^2 x_i^l}{\partial x_{j_1}^0 \partial x_{j_2}^0} &= \frac{\partial}{\partial x_{j_2}^0}[f'(y_i^l) \cdot \frac{\partial y_i^l}{\partial x_{j_1}^0}] \\
&= f'(y_i^l) \cdot \frac{\partial}{\partial x_{j_2}^0}(\frac{\partial y_i^l}{\partial x_{j_1}^0}) + \frac{\partial[f'(y_i^l)]}{\partial x_{j_2}^0} \cdot \frac{\partial y_i^l}{\partial x_{j_1}^0} \\
&= f'(y_i^l) \cdot \frac{\partial^2 y_i^l}{\partial x_{j_1}^0 \partial x_{j_2}^0} + f''(y_i^l) \cdot \frac{\partial y_i^l}{\partial x_{j_1}^0} \cdot \frac{\partial y_i^l}{\partial x_{j_2}^0}. \qquad (6.3)
\end{aligned}
$$

Notice that due to the linear dependence of y_i^l on x_k^{l-1}, we have

$$(\prod_{n=1}^{N} \frac{\partial}{\partial x_{j_n}^0}) y_i^l = \sum_k w_{ik}^l \cdot (\prod_{n=1}^{N} \frac{\partial}{\partial x_{j_n}^0}) x_k^{l-1}.$$

To generalize the above result to the N-th order derivatives, we define an n-partition of a set $\{j_1, \cdots, j_N\}$.

Definition. An n-partition of $\{j_1, \cdots, j_N\}$, which we denote by p_n^N, is a set of n non-empty subsets $p_{n,i}^N$ of $\{j_1, \cdots, j_N\}$ which satisfy:

(a). completeness: $\bigcup_{i=1}^n p_{n,i}^N = \{j_1, \cdots, j_N\}$;

(b). exclusiveness: $p_{n,i}^N \cap p_{n,j}^N = \emptyset$, for $j \neq i$.

We denote the set of all n-partitions of $\{j_1, \cdots, j_N\}$ by $P_n^N(j_1, \cdots, j_N)$.

Using the chain rule (6.21) with $g = x_i^l$ and $z_j^\tau = y_j^l$, $j = i$, we can write down the propagation formula for the N-th order derivatives

$$(\prod_{n=1}^{N} \frac{\partial}{\partial x_{j_n}^0}) x_i^l = \sum_{n=1}^{N} f^{(n)}(y_i^l) \sum_{p_n^N \in P_n^N(j_1, \cdots, j_N)} \prod_{k=1}^{n} [(\prod_{s \in p_{n,k}^N} \frac{\partial}{\partial x_s^0}) y_i^l]. \qquad (6.4)$$

6.2. Differentiation by Backward Propagation

Let us first see how the first order derivatives $\partial x_i^L / \partial x_j^l$ can be calculated recursively for $l = L-1, \cdots, 0$. By the chain rule of differentiation,

$$\begin{aligned}
\frac{\partial x_i^L}{\partial x_j^l} &= \sum_k \frac{\partial x_i^L}{\partial x_k^{l+1}} \cdot \frac{\partial x_k^{l+1}}{\partial x_j^l} \\
&= \sum_k \frac{\partial x_i^L}{\partial x_k^{l+1}} \cdot \frac{d x_k^{l+1}}{d y_k^{l+1}} \cdot \frac{\partial y_k^{l+1}}{\partial x_j^l} \\
&= \sum_k \frac{\partial x_i^L}{\partial x_k^{l+1}} \cdot f'(y_k^{l+1}) \cdot w_{kj}^{l+1}. \qquad (6.5)
\end{aligned}$$

Thus, starting with $\partial x_i^L / \partial x_k^{L-1} = f'(y_i^L) w_{ik}^L$, we are able to obtain $\partial x_i^L / \partial x_j^l$ recursively.

Also, it is possible to utilize first order derivatives obtained in (6.5) to derive higher order derivatives also recursively. For the second order derivatives,

$$\begin{aligned}
&\frac{\partial^2 x_i^L}{\partial x_j^l \partial x_k^l} \\
&= \sum_s \frac{\partial}{\partial x_j^l} (\frac{\partial x_i^L}{\partial x_s^{l+1}} \cdot f'(y_s^{l+1}) \cdot w_{sk}^{l+1}) \\
&= \sum_s \frac{\partial}{\partial x_j^l} (\frac{\partial x_i^L}{\partial x_s^{l+1}}) \cdot f'(y_s^{l+1}) \cdot w_{sk}^{l+1} + \sum_s \frac{\partial x_i^L}{\partial x_s^{l+1}} \cdot \frac{\partial f'(y_s^{l+1})}{\partial x_j^l} \cdot w_{sk}^{l+1} \\
&= \sum_{s,t} \frac{\partial x_t^{l+1}}{\partial x_j^l} \cdot \frac{\partial^2 x_i^L}{\partial x_s^{l+1} \partial x_t^{l+1}} \cdot f'(y_s^{l+1}) \cdot w_{sk}^{l+1} + \sum_s \frac{\partial x_i^L}{\partial x_s^{l+1}} \cdot f''(y_s^{l+1}) \cdot w_{sj}^{l+1} \cdot w_{sk}^{l+1} \\
&= \sum_{s,t} f'(y_t^{l+1}) \cdot w_{tj}^{l+1} \cdot \frac{\partial^2 x_i^L}{\partial x_s^{l+1} \partial x_t^{l+1}} \cdot f'(y_s^{l+1}) \cdot w_{sk}^{l+1} \\
&\quad + \sum_s \frac{\partial x_i^L}{\partial x_s^{l+1}} \cdot f''(y_s^{l+1}) \cdot w_{sj}^{l+1} \cdot w_{sk}^{l+1} \qquad (6.6)
\end{aligned}$$

with the initial condition $\partial^2 x_i^L / (\partial x_j^{L-1} \partial x_k^{L-1}) = f''(y_i^L) \cdot w_{ij}^L \cdot w_{ik}^L$.

Denote the set of all n-partitions (defined in previous section) of $\{j_1, \cdots, j_N\}$ as P_n^N. By the chain rule (6.21) with $g = x_i^L$ and $z_j^\tau = x_j^{l+1}$, $j = 1, \cdots, n_{l+1}$, the backward propagation formula for the N-th order derivatives is

$$(\prod_{n=1}^{N} \frac{\partial}{\partial x^{l}_{j_n}}) x_i^L = \sum_{n=1}^{N} \sum_{s_1,\cdots,s_n} (\prod_{k=1}^{n} \frac{\partial}{\partial x^{l+1}_{s_k}}) x_i^L \sum_{p_n^N \in P_n^N(j_1,\cdots,j_N)} [\prod_{t=1}^{n} f^{(m_{n,t}^N)}(y_{s_t}^{l+1}) \cdot (\prod_{u \in p_{n,t}^N} w_{s_t u}^{l+1})] \tag{6.7}$$

where $m_{n,t}^N$ is the number of elements in $p_{n,t}^N$.

6.3. Differentiating Partial Derivatives w.r.t. Weights

In this section, we give the formulas for calculating $\partial P/\partial w_{pq}^r$, where P is a partial derivative of an output variable w.r.t. input variables for a multilayer feedforward network, and w_{pq}^r is the synaptic weight connecting the q-th neuron in the $(r-1)$-th layer to the p-th neuron in the r-th layer. Also, we denote the weighted sum in and output of the i-th neuron in the l-th layer as y_i^l and x_i^l, respectively. And the i-th input variable is denoted x_i^0.

We use three different schemes to calculate $\partial P/\partial w_{pq}^r$. Forward and backward propagation methods can be derived directly from forward and backward propagation formulas for the partial derivatives. Furthermore, we propose a direct method which, in some case, has a more compact form.

The forward propagation formula for calculating $\partial P/\partial w_{pq}^r$ is obtained by differentiating Eq. (6.4),

$$\begin{aligned}
\frac{\partial}{\partial w_{pq}^r}(\prod_{n=1}^{N} \frac{\partial}{\partial x_{j_n}^0}) x_i^l &= \sum_{n=1}^{N} \frac{\partial [f^{(n)}(y_i^l)]}{w_{pq}^r} \sum_{p_n^N \in P_n^N(j_1,\cdots,j_N)} \prod_{k=1}^{n} [(\prod_{s \in p_{n,k}^N} \frac{\partial}{\partial x_s^0}) y_i^l] \\
&\quad + \sum_{n=1}^{N} f^{(n)}(y_i^l) \sum_{p_n^N \in P_n^N(j_1,\cdots,j_N)} \frac{\partial}{\partial w_{pq}^r} \prod_{k=1}^{n} [(\prod_{s \in p_{n,k}^N} \frac{\partial}{\partial x_s^0}) y_i^l] \\
&= \sum_{n=1}^{N} f^{(n+1)}(y_i^l) \frac{\partial y_i^l}{\partial w_{pq}^r} \sum_{p_n^N \in P_n^N(j_1,\cdots,j_N)} \prod_{k=1}^{n} [(\prod_{s \in p_{n,k}^N} \frac{\partial}{\partial x_s^0}) y_i^l] \\
&\quad + \sum_{n=1}^{N} f^{(n)}(y_i^l) \sum_{p_n^N \in P_n^N(j_1,\cdots,j_N)} \sum_{m=1}^{n} \frac{\frac{\partial}{\partial w_{pq}^r}[(\prod_{s \in p_{n,m}^N} \frac{\partial}{\partial x_s^0}) y_i^l]}{(\prod_{s \in p_{n,m}^N} \frac{\partial}{\partial x_s^0}) y_i^l} \\
&\quad \cdot \prod_{k=1}^{n} [(\prod_{s \in p_{n,k}^N} \frac{\partial}{\partial x_s^0}) y_i^l],
\end{aligned} \tag{6.8}$$

where $P_n^N(j_1,\cdots,j_N)$, p_n^N and $p_{n,k}^N$ are defined in Sec. II. We know that

$$\begin{aligned}
\frac{\partial}{\partial w_{pq}^r}[(\prod_{n=1}^N \frac{\partial}{\partial x_{j_n}^0})y_i^l] &= \frac{\partial}{\partial w_{pq}^r}\sum_k w_{ik}^l(\prod_{n=1}^N \frac{\partial}{\partial x_{j_n}^0})x_k^{l-1} \\
&= \delta_{rl}\delta_{pi}(\prod_{n=1}^N \frac{\partial}{\partial x_{j_n}^0})x_q^{l-1} + \sum_k w_{ik}^l \\
&\quad \cdot\frac{\partial}{\partial w_{pq}^r}[(\prod_{n=1}^N \frac{\partial}{\partial x_{j_n}^0})x_k^{l-1}], \qquad (6.9)
\end{aligned}$$

and

$$\frac{\partial y_i^l}{\partial w_{pq}^r} = \delta_{rl}\delta_{pi}x_q^{l-1} + \sum_k w_{ik}^l\frac{\partial x_k^{l-1}}{\partial w_{pq}^r}. \qquad (6.10)$$

The backward propagation formula can be obtained by differentiating Eq. (6.7),

$$\begin{aligned}
\frac{\partial}{\partial w_{pq}^r}[(\prod_{n=1}^N \frac{\partial}{\partial x_{j_n}^l})x_i^L] &= \sum_{n=1}^N\sum_{s_1,\cdots,s_n}\frac{\partial}{\partial w_{pq}^r}[(\prod_{k=1}^n \frac{\partial}{\partial x_{s_k}^{l+1}})x_i^L]\sum_{p_n^N\in P_n^N(j_1,\cdots,j_N)} \\
&\quad [\prod_{t=1}^n f^{(m_{n,t}^N)}(y_{s_t}^{l+1})(\prod_{u\in p_{n,t}^N} w_{s_tu}^{l+1})] + \sum_{n=1}^N\sum_{s_1,\cdots,s_n}(\prod_{k=1}^n\frac{\partial}{\partial x_{s_k}^{l+1}})x_i^L \\
&\quad \sum_{p_n^N\in P_n^N(j_1,\cdots,j_N)}\frac{\partial}{\partial w_{pq}^r}[\prod_{t=1}^n f^{(m_{n,t}^N)}(y_{s_t}^{l+1})\cdot(\prod_{u\in p_{n,t}^N} w_{s_tu}^{l+1})] \\
&= \sum_{n=1}^N\sum_{s_1,\cdots,s_n}\frac{\partial}{\partial w_{pq}^r}[(\prod_{k=1}^n \frac{\partial}{\partial x_{s_k}^{l+1}})x_i^L]\sum_{p_n^N\in P_n^N(j_1,\cdots,j_N)} \\
&\quad [\prod_{t=1}^n f^{(m_{n,t}^N)}(y_{s_t}^{l+1})(\prod_{u\in p_{n,t}^N} w_{s_tu}^{l+1})] + \sum_{n=1}^N\sum_{s_1,\cdots,s_n}(\prod_{k=1}^n\frac{\partial}{\partial x_{s_k}^{l+1}})x_i^L \\
&\quad \sum_{p_n^N\in P_n^N(j_1,\cdots,j_N)}\{\sum_{v=1}^n[\frac{f^{(m_{n,v}^N+1)}(y_{s_v}^{l+1})}{f^{(m_{n,v}^N)}(y_{s_v}^{l+1})}\cdot\frac{\partial y_{s_v}^{l+1}}{\partial w_{pq}^r} \\
&\quad + \sum_{w\in p_{n,v}^N}\frac{\delta_{s_vp}\delta_{l+1,r}\delta_{wq}}{w_{s_vw}^{l+1}}][\prod_{t=1}^n f^{(m_{n,t}^N)}(y_{s_t}^{l+1})(\prod_{u\in p_{n,t}^N} w_{s_tu}^{l+1})]\}(6.11)
\end{aligned}$$

Observing that $x^0_{j_n}$ is independent of w^r_{pq}, we are able to change their order of differentiation. This enable us to obtain $\partial P/\partial w^r_{pq}$ directly without propagation:

$$\begin{aligned}
\frac{\partial}{\partial w^r_{pq}}(\prod_{n=1}^{N}\frac{\partial}{\partial x^0_{j_n}})x^L_i &= (\prod_{n=1}^{N}\frac{\partial}{\partial x^0_{j_n}})(\frac{\partial x^L_i}{\partial w^r_{pq}}) \\
&= (\prod_{n=1}^{N}\frac{\partial}{\partial x^0_{j_n}})[\frac{\partial x^L_i}{\partial x^r_p}f'(y^r_p)x^{r-1}_q] \\
&= \sum_{c^N_3\in C^N_3(j_1,\cdots,j_N)}(\prod_{s\in c^N_{3,1}}\frac{\partial}{\partial x^0_s})\frac{\partial x^L_i}{\partial x^r_p} \\
&\quad \cdot(\prod_{t\in c^N_{3,2}}\frac{\partial}{\partial x^0_t})f'(y^r_p)\cdot(\prod_{u\in c^N_{3,3}}\frac{\partial}{\partial x^0_u})x^{r-1}_q, \qquad (6.12)
\end{aligned}$$

where c^N_3 is an *ordered* partition of $\{j_1,\cdots,j_N\}$ into 3 groups, $c^N_{3,1}$, $c^N_{3,2}$ and $c^N_{3,3}$, with 0 or more elements and $C^N_3(j_1,\cdots,j_N)$ is the set of all c^N_3. It is the sum of the products of three factors. The third factor can be obtained using Eq. (6.4). We rewrite the first factor using the chain rule (6.21) with $g = \partial x^L_i/\partial x^r_p$ and $z^r_j = x^r_j$, $j = 1,\cdots,n_r$,

$$\begin{aligned}
(\prod_{n=1}^{N}\frac{\partial}{\partial x^0_{j_n}})\frac{\partial x^L_i}{\partial x^r_p} &= \sum_{M=1}^{N}\sum_{k_1,\cdots,k_M}(\prod_{m=1}^{M}\frac{\partial}{\partial x^r_{k_m}})\frac{\partial x^L_i}{\partial x^r_p}\sum_{p^N_M\in P^N_M(j_1,\cdots,j_N)} \\
&\quad \prod_{m=1}^{M}[(\prod_{s\in p^N_{M,m}}\frac{\partial}{\partial x^0_s})x^r_{k_m}], \qquad (6.13)
\end{aligned}$$

where the first partial derivative can be obtained by Eq. (6.7) and the second can be obtained by Eq. (6.4). The second factor in (6.12) can be rewritten using the chain rule (6.21) with $g = f'(y^r_p)$ and $z^r_j = x^{r-1}_j$, $j = 1,\cdots,n_{r-1}$,

$$\begin{aligned}
(\prod_{n=1}^{N}\frac{\partial}{\partial x^0_{j_n}})f'(y^r_p) &= \sum_{M=1}^{N}\sum_{k_1,\cdots,k_M}f^{(M+1)}(y^r_p)(\prod_{m=1}^{M}w^r_{pk_m})\sum_{p^N_M\in P^N_M(j_1,\cdots,j_N)} \\
&\quad \prod_{m=1}^{M}[(\prod_{s\in p^N_{M,m}}\frac{\partial}{\partial x^0_s})x^{r-1}_{k_m}], \qquad (6.14)
\end{aligned}$$

where the partial derivative can be obtained by Eq. (6.4).

6.4. Estimating the covariance of the weights

In order to test the hypothesis that the weight on a certain connection is zero and the connection can be severed, we need the probability distribution of the weight. furthermore, these probability distributions of the weights determine those of the derivative of all orders of the multilayer feedforward network, which are needed to decide which are needed to decide which inputs and neurons can be pruned.

The statistical model that we use is the following:

$$y_t = f(x_t, w) + \epsilon_t, \; t = 1, \cdots, T, \tag{6.15}$$

where (x_t, y_t) is the t-th training pair, w is the vector of all the weights, $\{\epsilon_t\}$ are iid normal with mean 0 and covariance V, and $f(x_t, w)$ is the output of the network with weights w for the input x_t.

Rewrite (6.15) as a vector equation

$$y = F(x, w) + \epsilon, \tag{6.16}$$

for $y = [y_1', \cdots, y_T']'$, $F(x, w) = [f(x_1, w)', \cdots, f(x_T, w)']'$ and $\epsilon = [\epsilon_1', \cdots, \epsilon_T']'$. Let w^* denote the weight vector obtained from training. We linearize F about w^* and approximate (6.16) by

$$y = F(x, w^*) + \nabla_w F(x, w^*)(w - w^*) + \epsilon.$$

Note that $\nabla_w F(x, w^*)$ can be obtained by the standard backpropagation. The estimation for w for a different realization of ϵ and thus y is the linear regression for the linear model

$$y - F(x, w^*) = \nabla_w F(x, w^*)(w - w^*) + \epsilon,$$

which yields the estimate

$$\hat{w} = w^* + [(\nabla_w F(x, w^*))'(\nabla_w F(x, w^*))]^{-1}(\nabla_w F(x, w^*))'(y - F(x, w^*)).$$

Hence the covariance of $\hat{w}$ is

$$\begin{aligned} E[(\hat{w} - w)(\hat{w} - w)'] &= [(\nabla_w F(x, w^*))'(\nabla_w F(x, w^*))]^{-1}(\nabla_w F(x, w^*))'V \\ &\quad (\nabla_w F(x, w^*))[(\nabla_w F(x, w^*))'(\nabla_w F(x, w^*))]^{-1}.(6.17) \end{aligned}$$

Since w^* is one of those $\hat{w}$, we conclude that w^* in repeated sampling has covariance equal to the right side of (6.17).

It goes without saying that the above formula for the covariance of w^* is valid only if the errors ϵ are so sufficiently small that the foregoing linearization is justified.

6.5. Estimating the covariance of the partial derivatives of a multilayer feedforward network

The partial derivatives of a multilayer feedforward network are functions of the weight vector w of the network. The variability of the estimate $\hat{w}$ of w induces that of the partial derivatives of the network with weights set equal to $\hat{w}$.

Denoting by $D(w)$ the partial derivative under consideration at a given input vector of the network with respect to weights w, we linearize D about w^* as follows:

$$D(w) \cong D(w^*) + \nabla_w D(w^*)(w - w^*). \tag{6.18}$$

In particular, at an estimate $\hat{w}$ of w,

$$D(\hat{w}) \cong D(w^*) + \nabla_w D(w^*)(\hat{w} - w^*). \tag{6.19}$$

Substracting (6.18) from (6.19) yields

$$\begin{aligned} D(\hat{w}) - D(w) &\cong \nabla_w D(w^*)(\hat{w} - w), \\ E[(D(\hat{w}) - D(w))^2] &\cong \nabla_w D(w^*) E[(\hat{w} - w)(\hat{w} - w)'](\nabla_w D(w^*))'. \end{aligned} \tag{6.20}$$

We stress that the symbol D denotes a certain partial derivative at a given input vector throughout the foregoing calculation.

6.6. Pruning weights and input variables by testing hypotheses

Let us briefly review the method of testing the null hypothesis H_0 that the population mean μ is zero versus the alternative hypothesis H_1 that it is not. If a statistic x, whose mean and variance in repeated sampling are μ and s^2 respectively, is obtained, the z-statistic z under H_0, which is the number of standard deviations x/s away from $\mu = 0$, is usually used for test the hypothesis H_0. The observed significance level is the chance of getting a test statistic as extreme as or more extreme than the observed one. The smaller this chance is, the stronger the evidence against the null.

For instance, if $z = 2.00$, then the observed significance level is 4.55%. This level is usually accepted as sufficiently small in practice and a z-statistic greater than or equal to 2.00 is viewed as sufficient evidence to reject the null hypothesis.

The statistic that we will use to test the hypothesis that a synaptic weight w_{ij}^l is zero and thus the connection should be pruned is the component of w^* that is an estimate of the synaptic weight. Given the variance of the component of w^* as computed in the preceding section, we can evaluate the z-statistic for it. If the z-statistic is less than 2.00, we suspect that corresponding component of w is actually zero and the corresponding connection should be pruned.

Similarly, the statistic for testing the hypothesis that a derivative $D(w)$ at an input vector is zero is $D(w^*)$ evaluated at the same input vector. If the z-statistic $D(w^*)/(\text{the standard deviation of } D(w^*))$ is less than 2.00 at the input vector, we suspect that $D(w)$ is zero there.

6.7. A Numerical Example

We will now illustrate how we can prune weights and input variables (or neurons) by an example. We consider a (2:2:1) feedforward network with an identity activation function for the output neuron. Denoting the 2 input variables by x and y, the network is trained to approximate the function $g(x,y) = (1 + \sin 2\pi x)/2$ over the region $[0,1) \times [0,1)$. The training data is obtained by first selecting 400 points at random from the region and evaluating $g(x,y)$ at each point. Then 400 statistically independent normal random numbers with mean zero and standard deviation 0.005 are generated. Each is added to one of the 400 values of $g(x,y)$. The 400 pairs of $((x,y), g(x,y))$ are the training data.

Training by conjugate gradient method brought the MSE down to 2.66×10^{-5} consistent with the standard deviation (0.005) of the additive error in the training output data. The weights w^* are

$$\begin{aligned} [w_{ij}^{1*}] &= \begin{bmatrix} 0.5946 & -1.1836 & 0.0001 \\ 1.2442 & -2.4851 & 0.0002 \end{bmatrix}, \\ [w_{ij}^{2*}] &= \begin{bmatrix} 0.5152 & -8.5174 & 5.3379 \end{bmatrix}. \end{aligned}$$

The standard deviations of w^*, which were calculated by the formula (6.17), are

$$\begin{aligned} [s_{ij}^{1}] &= \begin{bmatrix} 0.1363 & 0.2738 & 0.0005 \\ 0.0882 & 0.1764 & 0.0010 \end{bmatrix}, \\ [s_{ij}^{2}] &= \begin{bmatrix} 0.0146 & 1.1031 & 1.8380 \end{bmatrix}. \end{aligned}$$

The z-statistics for w are then

$$\begin{aligned} [w_{ij}^{1*}/s_{ij}^{1}] &= \begin{bmatrix} 4.3625 & -4.3224 & 0.2017 \\ 14.1022 & -14.0878 & 0.2038 \end{bmatrix}, \\ [w_{ij}^{2*}/s_{ij}^{2}] &= \begin{bmatrix} 35.2552 & -7.7212 & 2.9042 \end{bmatrix}. \end{aligned}$$

We notice that the above z-statistics indicate that w_{13} and w_{23} are possibly prunable. In fact they are the weights leading out of the input variable y, which actually have no effect on the function $g(x,y) = (1 + \sin 2\pi x)/2$ that the network is intended to approximate.

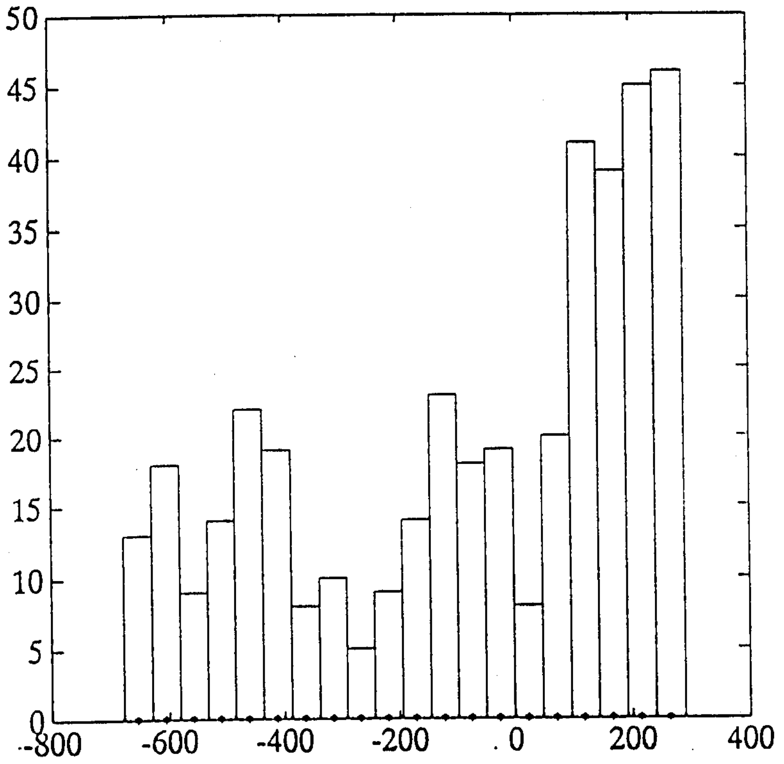


Figure 6.1: z-statistics for $\partial\mathcal{O}/\partial x$

The z-statistics for the first order derivatives $\partial\mathcal{O}/\partial x$ and $\partial\mathcal{O}/\partial y$ at each of the 400 input vectors were also calculated, where $\mathcal{O}$ denotes the output of the feedforward network. The histograms for the z-statistics for $\partial\mathcal{O}/\partial x$ and $\partial\mathcal{O}/\partial y$ are given in Figs. 6.1 and 6.2 respectively. The z-statistics for $\partial\mathcal{O}/\partial y$ are distributed between -0.2 and 0.15, suggesting that the input variable y is perhaps prunable.

6.8. Conclusions

If its activation functions are analytic, a multilayer feedforward neural network is also analytic, whose Taylor series expansions converge to it. The backward and forward propagation methods of calculating its derivatives of all orders facilitate efficiently such series expansions. These series expansions provide ample intuitive interpretation of the functional relationships hidden in the training data.

The covariances of the weights and the derivatives allow the determination of the confidence regions and the testing of the hypotheses that a weight, a neuron or an input is prunable. They reflect the sensitivity of the weights and derivatives

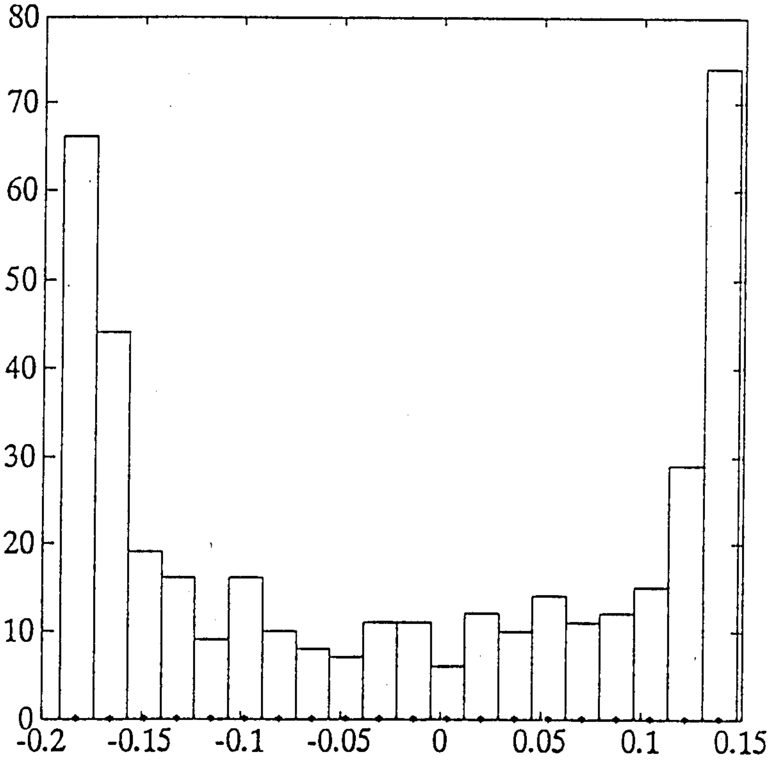


Figure 6.2: z-statistics for $\partial\mathcal{O}/\partial y$

w.r.t. the noises in the training data and remove the subjectivity in network pruning.

We are restricted to multilayer feedforward neural networks in this paper. Nevertheless, the results can be extended to recurrent neural networks of all types. The extensions will soon be reported in a forthcoming paper of the authors.

6.9. Appendix: A Chain Rule for Differentiating Multilayer Feedforward Networks

Suppose $\{z^r_1, \cdots, z^r_n\}$ are either a subset of $\{x^r_1, \cdots, x^r_{n_r}\}$ or a subset of $\{y^r_1, \cdots, y^r_{n_r}\}$ and g is an N-th order differentiable function of $\{z^r_1, \cdots, z^r_n\}$. Given $r \geq l$, the N-th order derivative $(\prod_{n=1}^N \partial/\partial x^l_{j_n})g$ can be expressed in terms of the derivatives of g with respect to $\{z^r_1, \cdots, z^r_{n_r}\}$ and the derivatives of z^r_i (for i from 1 to n_r) with respect to $x^l_{j_n}$ as follows:

$$(\prod_{n=1}^{N} \frac{\partial}{\partial x^l_{j_n}})g = \sum_{n=1}^{N} \sum_{s_1,\cdots,s_n} (\prod_{k=1}^{n} \frac{\partial}{\partial z^r_{s_k}})g \cdot \sum_{p^N_n \in P^N_n(j_1,\cdots,j_N)} [\prod_{t=1}^{n} (\prod_{u \in p^N_{n,t}} \frac{\partial}{\partial x^l_u}) z^r_{s_t}]. \qquad (6.21)$$

We refer to it as the chain rule for multilayer feedforward networks. Eq. (6.21) can be proved by mathematical induction. We first note that:

$$\begin{aligned} P^{N+1}_{n+1}(j_1, \cdots, j_{N+1}) \; = \; & \{p^N_n \cup \{\{j_{N+1}\}\} | p^N_n \in P^N_n(j_1, \cdots, j_N)\} \\ & \cup \{p^{N,1}_n | p^N_n \in P^N_n(j_1, \cdots, j_N)\} \\ & \cup \{p^{N,2}_n | p^N_n \in P^N_n(j_1, \cdots, j_N)\} \\ & \cdots \\ & \cup \{p^{N,n}_n | p^N_n \in P^N_n(j_1, \cdots, j_N)\}, \end{aligned} \qquad (6.22)$$

where

$$\begin{aligned} p^{N,m}_n & \triangleq \{p^{N,m}_{n,1}, \cdots, p^{N,m}_{n,n}\}, \\ p^{N,m}_{n,k} & \triangleq \begin{cases} p^N_{n,k} \cup \{j_{N+1}\}, & \text{if } k = m; \\ p^N_{n,k}, & \text{otherwise.} \end{cases} \end{aligned}$$

Suppose (6.21) holds, then

$$\frac{\partial}{\partial x^l_{j_{N+1}}} (\prod_{n=1}^{N} \frac{\partial}{\partial x^l_{j_n}})g$$

$$\begin{aligned}
&= \sum_{n=1}^{N} \sum_{s_1,\cdots,s_n} \frac{\partial}{\partial x^l_{j_{N+1}}} (\prod_{k=1}^{n} \frac{\partial}{\partial z^r_{s_k}}) g \cdot \sum_{p^N_n \in P^N_n(j_1,\cdots,j_N)} [\prod_{t=1}^{n} (\prod_{u \in p^N_{n,t}} \frac{\partial}{\partial x^l_u}) z^r_{s_t}] \\
&\quad + \sum_{n=1}^{N} \sum_{s_1,\cdots,s_n} (\prod_{k=1}^{n} \frac{\partial}{\partial z^r_{s_k}}) g \cdot \sum_{p^N_n \in P^N_n(j_1,\cdots,j_N)} \frac{\partial}{\partial x^l_{j_{N+1}}} [\prod_{t=1}^{n} (\prod_{u \in p^N_{n,t}} \frac{\partial}{\partial x^l_u}) z^r_{s_t}] \\
&= \sum_{n=1}^{N} \sum_{s_1,\cdots,s_{n+1}} \frac{\partial z^r_{s_{n+1}}}{\partial x^l_{j_{N+1}}} (\prod_{k=1}^{n+1} \frac{\partial}{\partial z^r_{s_k}}) g \cdot \sum_{p^N_n \in P^N_n(j_1,\cdots,j_N)} [\prod_{t=1}^{n} (\prod_{u \in p^N_{n,t}} \frac{\partial}{\partial x^l_u}) z^r_{s_t}] \\
&\quad + \sum_{n=1}^{N} \sum_{s_1,\cdots,s_n} (\prod_{k=1}^{n} \frac{\partial}{\partial z^r_{s_k}}) g \cdot \sum_{p^N_n \in P^N_n(j_1,\cdots,j_N)} \sum_{m=1}^{n} [\prod_{t=1}^{n} (\prod_{u \in p^{N,m}_{n,t}} \frac{\partial}{\partial x^l_u}) z^r_{s_t}] \\
&= \sum_{n=1}^{N+1} \sum_{s_1,\cdots,s_n} (\prod_{k=1}^{n} \frac{\partial}{\partial z^r_{s_k}}) g \cdot \sum_{p^{N+1}_n \in P^{N+1}_n(j_1,\cdots,j_{N+1})} [\prod_{t=1}^{n} (\prod_{u \in p^{N+1}_{n,t}} \frac{\partial}{\partial x^l_u}) z^r_{s_t}]
\end{aligned}$$

which completes proof.

7. Conclusions

The fundation of a very powerful new approach to optimal filtering has been laid down. As opposed to the conventional theories, this approach is synthetic in nature. It synthesizes realizations of the signal and measurement processes into a filter. If a mathematical model of the signal and measurement processes is available, realizations of them can be simulated on a computer. If not, actual measurements of these processes can be used, eliminating the need to construct a mathematical model of these process.

Mathematical assumptions such as Markov property, linear dynamics, Gaussian distribution, and additive noise are not required in this new approach. It does not derive equations or formulas, but synthesizes a filter through training and selecting recurrent neural networks (RNNs). The selected RNN after training, which is called a neural filter, can be made as close to an optimal filter in performance as desired.

The massively parallel structures of the RNNs allow real-time implementation of the neural filters in many computation-intensive applications such as in image and video processing. The availability of analog and digital neural computing VLSI chips makes this real-time implementation simple and affordable.

There are still three problem areas which remain to be explored further. The first is adaptive neural filtering. The initial results reported in Chapter 5 are exciting, indicating that a lot more can be done. The neural network approach is a very natural one to treat adaptive filtering, due to the learning capability of neural networks.

The second problem area is the selection or determination of an optimal RNN architecture for a given application. Growing and pruning the neurons and connections are two important issues to be studied.

The last, but not the least, problem area is the training algorithms of the RNNs. Existing algorithms are slow and need large number of repeated trainings with different initial weights/parameters for the RNN to avoid a poor local minimum of the training criterion. Those more reliable algorithms also require a very large memory (e.g. 128MB or more) on the computer. Therefore, better algorithms for training RNNs are highly desirable.

8. REFERENCES

[1] B. D. O. Anderson and J. B. Moore. *Optimal Filtering.* Prentice-Hall, Inc, Englewood Cliffs, N.J., 1979.

[2] R. S. Bucy and K. D. Senne. Digital synthesis of non-linear filters. *Automatica*, 7:287–298, 1971.

[3] G. Cybenko. Continuous valued neural networks with two hidden layers are sufficient. Technical report, Department of Computer Science, Tufts University, Medford, MA, 1988.

[4] N. R. Draper and H. Smith. *Applied Regression Analysis.* John Wiley and Sons, New York, 1981.

[5] J. L. Elman. Finding structure in time. Technical Report CRL 8801, Center for Research in Language, University of California, San Diego, 1988.

[6] K. Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2:183–192, 1989.

[7] R. Hecht-Nielsen. *Neurocomputing.* Addison-Wesley Publishing Company, New York, NY, 1990.

[8] J. Hertz, A. Krogh, and R. G. Palmer. *Introduction to the Theory of Neural Computation.* Addison Wesley, Redwood City, CA, 1991.

[9] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

[10] K. Hornik, M. Stinchcombe, and H. White. Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural Networks*, 3:551–560, 1990.

[11] A. H. Jazwinski. *Stochastic Processes and Filtering Theory.* Academic Press, 1970.

[12] M. I. Jordan. Attractor dynamics and parallelism in a connectionist sequential machine. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, pages 531–546, Amherst, 1986. ErlBaum.

[13] B. Kosko, editor. *Neural Networks for Signal Processing.* Prentice Hall, 1992.

[14] V. Krishnan. *Nonlinear Filtering and Smoothing: An Introduction to Martingales, Stochastic Integrals and Estimation.* John Wiley & Sons, 1984.

[15] R. S. Liptser and A. N. Shiryayev. *Statistics of Random Processes I: general theory.* Springer-Verlag, New York, 1977.

[16] J. T.-H. Lo. Optimal estimation for the satellite attitude using star tracker measurements. *Automatica*, 22:477–482, 1986.

[17] J. T.-H. Lo and S.-K. Ng. Optimal functional expansion for estimation for counting observations. *IEEE Transactions on Information Theory*, IT-33:21–35, 1987.

[18] D. P. Morgan and C. L. Scofield. *Neural Networks and Speech Processing.* Kluwer Academic Publishers, Norwell, Massachusetts, 1991.

[19] D. B. Parker. Learning logic. Technical Report TR-47, Center for Computational Research in Economics and Managemant Science, Massachusetts Institute of Technology, Cambridge, MA, 1985.

[20] B. A. Pearlmutter. Learning state space trajectories in recurrent neural networks. *Neural Computation*, 1:263–269, 1989.

[21] G. V. Puskorious and L. A. Feldkamp. Recurrent network training with the decoupled extended Kalman filter algorithm. In *Proceedings of the 1992 SPIE Conference on the Science of Artificial Neural Networks*, Orlando, Florida, 1992.

[22] D. E. Rumelhart, G. E. Hinton, and P. J. Williams. Learning internal representation by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing.* MIT Press, 1986.

[23] E. Sanchez-Sinencio and C. Lau, editors. *Artificial Neural Networks.* IEEE Press, 1992.

[24] H. W. Sorenson, editor. *Kalman Filtering: Theory and Application.* IEEE Press, 1985.

[25] S. I. Sudharsanan and M. K. Sundareshan. Maximum a posteriori state estimation: A neural processing algorithm. In *Proceedings of the 28th Conference on Decision and Control*, pages 1805–1806, 1989.

[26] P. J. Werbos. *Beyond regression: New Tools for Prediction and Analysis in the Behavioral Sciences.* PhD thesis, Harvard University, Cambridge, Massachussetts, 1974.

[27] P. J. Werbos. Generalization of backpropagation with application to a recurrent gas market model. *Neural Networks*, 1:339–356, 1988.

[28] D. A. White and D. A. Sofge. *Handbook of Intelligent Control.* Van Nostrand Reinhold, New York, NY, 1992.

☆U.S. GOVERNMENT PRINTING OFFICE: 1995-610-126-20049

MISSION
OF
ROME LABORATORY

Mission. The mission of Rome Laboratory is to advance the science and technologies of command, control, communications and intelligence and to transition them into systems to meet customer needs. To achieve this, Rome Lab:

a. Conducts vigorous research, development and test programs in all applicable technologies;

b. Transitions technology to current and future systems to improve operational capability, readiness, and supportability;

c. Provides a full range of technical support to Air Force Materiel Command product centers and other Air Force organizations;

d. Promotes transfer of technology to the private sector;

e. Maintains leading edge technological expertise in the areas of surveillance, communications, command and control, intelligence, reliability science, electro-magnetic technology, photonics, signal processing, and computational science.

The thrust areas of technical competence include: Surveillance, Communications, Command and Control, Intelligence, Signal Processing, Computer Science and Technology, Electromagnetic Technology, Photonics and Reliability Sciences.